University of Zurich[UZH]

Department of Informatics

Interactive Multiresolution and Multiscale Visualization of Large Volume Data

A dissertation submitted to the Faculty of Economics, Business Administration and Information Technology of the University of Zurich

for the degree of
Doctor of Science

by
Susanne Suter
from Mettmenstetten, Switzerland

Accepted on the recommendation of

Prof. Dr. Renato Pajarola
Prof. Dr. Christoph Zollikofer
Prof. Dr. Antonio Susín Sánchez

2013

The Faculty of Economics, Business Administration and Information Technology of the University of Zurich herewith permits the publication of the aforementioned dissertation without expressing any opinion on the views contained therein.

Zurich, April 3, 2013

The head of the Ph.D. program in informatics: Prof. Abraham Bernstein, Ph.D.

KURZFASSUNG

Interaktive Visualisierung und Analyse von komplexen Volumendatensätzen ist eine permanente Herausforderung. Datenakquisitionsgeräte produzieren heutzutage Daten im dreistelligen Gigabytebereich und sind normalerweise den verfügbaren Visualisierungssystemen einen Schritt voraus. Das bedeutet, dass die darzustellende Datenmenge um einiges grösser ist als was die verfügbare Computergrafikhardware verarbeiten kann. Wir gehen diese Herausforderung im Kontext von hochmodernen Visualisierungssystemen basierend auf Mehrfachauflösungen an (multiresolution volume visualization). Insbesondere benutzen wir dazu ein mathematisches Bezugssystem, welches die Daten in Basen und Koeffizienten zerlegt, um drei Herausforderungen gleichzeitig zu lösen: (a) Extraktion von relevanten Merkmalen innerhalb eines Datensatzes, (b) Datenreduktion und (c) direkte Visualisierung ausgehend von den Koeffizienten der zerlegten Daten.

Diese Doktorarbeit beinhaltet eine ausführliche Analyse von modernen Datennäherungsverfahren (data approximation approaches) und wie diese für die interaktive Visualisierung von Volumendaten mit spezifischen Merkmalen verwendet werden können. Daten werden oft mittels kompakter Datenrepräsentation angenähert oder reduziert, wodurch eine kleinere Menge an Koeffizienten als Rohdaten gebraucht wird. In dieser Arbeit wurde eine Erweiterung der Singulärwertzerlegung – Tensor Approximation (TA) – als kompakte Datenrepräsentation gewählt. TA besteht aus zwei Teilen: (1) Die *Tensordekomposition* zerlegt die Daten in Basen und Koeffizienten und (2) die *Tensorrekonstruktion* fügt die zerlegten Daten für die Visualisierung wieder zusammen.

Aufbauend auf diesem Konzept verknüpfen wir multiresolution Visualisierung und multiscale[1] Merkmalsektraktion mit Tensor Approximation. Die zwei Achsen der TA-Basen können je für multiresolution und multiscale Visualisierung verwendet werden. Die Eigenschaften der vertikalen TA-Basenachse eignen sich für die Modellierung moderner multiresolution Visualisierung. Dabei werden einerseits verschiedene Detaillierungsgrade mit unterschiedlich hoch aufgelösten Rohdaten repräsentiert und anderseits kleinere Datenportionen geladen, welche auf der Grafikkarte Platz haben. Beide diese Datentransformationen können direkt in den TA-Basen anstatt in den Rohdaten vorgenommen werden. Die horizontale Achse der TA-Basen erlaubt es über den sogenannten Tensorrang die Daten für unterschiedliche Merkmalsskalen zu rekonstruieren. Die Rekonstruktion mit einer kleinen Anzahl Rängen entspricht einer tiefrangigen Datenanäherung (viele Details entfernt) während die Rekonstruktion mit einer grossen Anzahl von Rängen fast mit den Rohdaten übereinstimmt. Zudem wurde eine Metrik für die Visualisierung von unterschiedlichen Merkmalsskalen entwickelt, um im Visualisisierungssystem automatisch die Auflösung und die Skala der vorhandenen Merkmale zu steuern. Diese Merkmalsskala-Metrik kann ein Benutzer interaktiv im Visualisierungssystem anpassen.

Dank der kompakten Datenrepräsentation via TA konnte eine bedeutsame Kompressionsrate erreicht werden (15 Prozent der ursprünglichen Anzahl Datenelemente). Diese Datenkompression hält die Speicherkosten tief und steigert gleichzeitig die Interaktivitätsrate von Visualisierungssystemen. Die interaktive Visualisierung wurde zudem dadurch ermöglicht, dass die Rekonstruktion parallel und direkt auf dem Grafikprozessor ausgeführt wird.

Die Brauchbarkeit der interaktiven Visualisierung von multiscale Merkmalsextraktion und Mehrfachuflösungen wurden mit zwei Tensor-basierten multiresolution Modellen getestet: (1) Ein TA Modell für lokale Datenblöcke und (2) ein Modell mit globalen TA Basen. Beide TA Volumenmodelle benutzen die *vmmlib* Tensorklassen, welche spezifisch für diese Doktorarbeit entwickelt wurden. Die Visualisierungsmodelle wurden mit bis zu 34GB grossen microCT und Phasenkontrast Synchrotronstomographie Datensätzen getestet. Es werden qualitative (visuelle) und quantitative Vergleiche von TA und anderen modernen Kompressionsverfahren wie Wavelets gezeigt.

Zum Schluss werden die Kernpunkte herausgehoben, welche es ermöglichen Tensor Approximation als vereinheitlichtes Bezugssystem zu benutzen, um multiscale Merkmalsextraktion und Mehrfachauflösungen in einem Visualisierungssystem zu modellieren. Dazu werden die erreichten Resultate und mögliche zukünftige Erweiterungen diskutiert.

[1] Mit *multiscale* Merkmalsektraktion ist gemeint, dass im Volumen vorhandene Merkmale auf unterschiedlichen räumlichen Skalen sichtbar gemacht werden können.

ABSTRACT

Interactive visualization and analysis of large and complex volume data is an ongoing challenge. Data acquisition tools produce hundreds of Gigabytes of data and are one step ahead of visualization and analysis tools. Therefore, the amount of data to be rendered is typically beyond the limits of current computer and graphics hardware performance. We tackle this challenge in the context of state-of-the-art out-of-core multiresolution volume rendering systems by using a common mathematical framework (a) to extract relevant features from these large datasets, (b) to reduce and compress the actual amount of data, and (c) to directly render/visualize the data from the framework coefficients.

This thesis includes an extended state-of-the-art analysis of data approximation approaches and how they can be applied to interactive volume visualization and used for feature extraction. Data is often approximated or reduced by using compact data representations, which require fewer coefficients than the original dataset. In this thesis, the higher-order extension of the matrix singular value decomposition summarized under the term *tensor approximation* (TA) was chosen as compact data representation. Tensor approximation consists of two parts: (1) *tensor decomposition*, usually an offline process, to compute the bases and coefficients, and (2) *tensor reconstruction*, typically a fast real-time process that inverts the decomposition back to the original data during visualization.

From these basic concepts, we derive how *multiresolution volume visualization* and *multiscale feature extraction* are linked to the tensor approximation framework. The two axes of the TA bases were chosen as handles for multiresolu-

tion and multiscale visualization. The properties along the vertical axis of the TA bases match well the needs of state-of-the-art out-of-core multiresolution volume visualization, where different levels of detail are represented by coarser or higher resolution representations of the same dataset and portions of the original dataset are loaded on demand in the desired resolution. Thus, the vertical axis of the TA bases is used for spatial selectivity and subsampling of data blocks. The horizontal axis of the TA bases makes it possible to reconstruct the dataset at multiple feature scales through the so-called tensor rank. Choosing only a few ranks corresponds to a low-rank approximation (many details removed) and choosing many ranks corresponds to an approximation more closely matching the original. Furthermore, a feature scale metric was developed to automatically select a feature scale and a resolution for the final reconstruction. In this scenario, the user selects a desired feature scale for the approximated data, which is then used by the visualization system to automatically define the resolution and the feature scale for the current view on the dataset.

Thanks to the compact data representation by TA, a significant data compression (15 percent of the original data elements) was achieved, which keeps the storage costs low and boosts the interactive visualization. The interactive visualization is moreover accelerated by using GPU-based tensor reconstruction approaches.

The viability of interactive multiscale and multiresolution volume visualization is tested with different TA volume visualization frameworks: (1) a simple bricked TA multiresolution, and (2) a TA multiresolution framework that uses global tensor bases. Both TA frameworks build on the *vmmlib tensor classes*, which were specifically developed for this thesis. For the testing, large volume datasets from micro-computed tomography (microCT) and phase-contrast synchrotron tomography (pcST) that range up to 34 Gigabytes were acquired. We show visual as well as computational comparisons to state-of-the-art approaches such as wavelet transform.

We conclude by pointing out the tensor approximation framework to be a unified framework for interactive multiscale and multiresolution volume visualization systems, which directly controls data approximation in terms of feature scale and multiple levels of detail. To wrap up, we discuss the achieved results and outline possible future work directions.

ACKNOWLEDGEMENTS

I would like to acknowledge a whole network of people who accompanied, supported and inspired me during my thesis. First of all, I would like to express my gratitude to my main supervisor, Renato Pajarola. He supported me in many ways and triggered the development of a new volume rendering framework always being open to explore new paths. He, furthermore, supported various conference attendances and exchanges with other research groups which I enjoyed a lot. I thank Renato for giving me the opportunity to pursue my PhD at his lab. Special thanks go also to my co-supervisor, Christoph Zollikofer, who was the driving force to initiate my thesis project. I thank Christoph, moreover, for his support in personal and research life and for giving me the opportunity to have access to various data acquisition devices and to be part of an inspiring research environment.

During my thesis, I had the chance to visit another research lab and to interact with several domain experts. I send many thanks to the VicLab in Sardinia, Italy, for sharing research ideas and a research project with me. Thus, special thanks go to Enrico Gobbetti and his group for hosting me and for a successful ongoing research exchange and collaboration. I thank M. Gopi for his interactions on the SVD and his brainstorming on tensor approximations. I thank Shmuel Friedland for his interactions on tensor math. I thank Takeru Akasawa Sensei and Mieko San for the time spent at the Syrian excavation site. I thank Jose for the shared intensive "extreme" programming sessions, the hard-working paper-submission periods and his optimism to explore research areas. I thank Jody for his support in programming and for his aptitude in spotting unclear areas in the thesis. I thank

Marcia for unforgettable conference attendances and for “woman” talks.

I was part of two research groups during my thesis. I thank all the previous and current group members of the VMMLab and the MorphoLab of the University of Zurich. Besides an inspiring research environment, I also met a great atmosphere, and long-lasting as well as pulsating discussions during numerous aperos, hundreds of coffee breaks (including delicious sweets), and several group dinners or group events. In particular, I thank all my office mates, and Jonas for pushing me to contribute to vmmlib, Max for supporting me in any rendering issues, and Rafa for taking over the tensor volume rendering project.

I additionally enjoyed a network of peers who supported me in my academic development and in my personal life. Many thanks go to Stefania Leone and Christina Christina Papageorgopoulou, two valuable friends and companions on the research track. Many thanks go also to all members of the peer mentoring group CareerElixier. They all gave me power to continue following my passion in research and shared many fruitful discussions with me.

My greatest thanks go to my parents for their unlimited love and for their support of my personal life and my PhD. Special thanks go, moreover, to Hans Schwarzenbach who always was excited about my research projects and who kept sending me related news articles. My thanks go also to my sisters, Brigitte, Barbara, and Ursina, for always being there, and my husband for wonderful times spent together, for caring and for giving me the time to work on my thesis.

This work was funded in parts by the Forschungskredit of the University of Zürich and a Swiss National Science Foundation grant (project n°200021_132521). I acknowledge furthermore several publicly available tools and datasets, which were valuable sources for the development of this thesis. Namely, this are the OsiriX (`http://www.osirix-viewer.com`) and the voreen (`http://www.voreen.org`) visualization frameworks that were used to produce some of the images, the color brewer (`http://www.colorbrewer.org`), which was used to borrow great color schemes, and volume datasets as declared in App. A.

Finally, I wish to thank all the reviewers and readers of my thesis, and Toni Susín for taking the time and enthusiasm to be co-supervisor of my thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

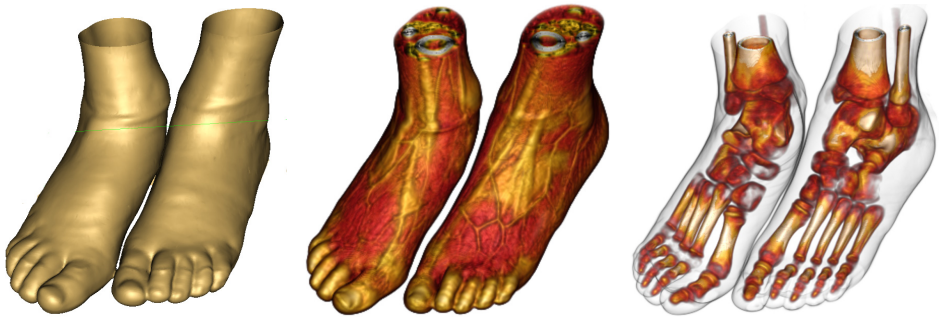
LIST OF ALGORITHMS

C H A P T E R 1

INTRODUCTION

1.1 Large Volume Data Visualization

Today, 3D visualization has become an integral part of many research disciplines like biology [Muller, 2002; Friis et al., 2007; Mizutani et al., 2007; Cano et al., 2008; Tafforeau and Smith, 2008; Hadwiger et al., 2012], physics [Springel et al., 2008; Stadel et al., 2009], geography [Geller, 2007; Pajarola and Gobbetti, 2007] and other disciplines [Duchaineau et al., 2000; Gobbetti and Marton, 2005; Cignoni et al., 2003]. In particular, 3D visualization allows users/researchers to inspect and explore datasets visually. 3D direct volume visualization systems are nowadays a standard tool to analyze, explore and inspect large amount of data. Direct volume visualization approaches are desired in order to visualize arbitrary cross-sections through a volume and to display transparent areas of an object. This can be achieved by so-called *direct volume rendering* (DVR), which displays the full volumetric data rather than only an iso-surface as usually visualized by mesh approaches, e.g., Fig. 1.1. In particular, the *ray casting* algorithm has become the standard DVR implementation since it produces high-quality images and its high computing demands are nowadays addressable by running parallel programs directly on the GPU. However, 3D data acquisition devices are typically still one step ahead of 3D visualization systems and produce datasets exceeding the graphics unit's memory. Moreover, even if the acquired datasets fit the memory, computing the DVR integrals still remains too costly for real-time rendering.



(a) (b) (c)

Figure 1.1: *Rendering approaches shown on a medical dataset (OsiriX database): (a) iso-surface rendering, (b-c) direct volume rendering showing different/transparent tissues.*

There are basically two ways to address the bottleneck of visualizing datasets that are larger than the available memory and computing resources: (a) The actual amount of data can be reduced prior to rendering/visualization, or (b) the actual rendering efficiency can be optimized. Data reduction can be achieved by a series

of transformations/actions performed on the data. Such actions are the decomposition of the data into a different representation, the truncation/thresholding of insignificant coefficients, as well as the quantization and encoding of the entities. The rendering efficiency, for instance, can be improved by using parallel algorithms, optimizing or approximating existing algorithms, using hardware acceleration and/or GPU-based implementations, downgrading the resolution dependent on the viewer, or employing a data management that returns data on demand. Typically the data is organized/structured during a preprocessing routine into a convenient compressed data format.

Since the graphics memory is limited and data transfer from CPU to GPU is time consuming, it is a major goal to reduce the size of the datasets as much as possible. Data reduction is often achieved by using *compact data representation models*, which require fewer bits for encoding than the original data. In the literature, this concept is known as *compression-domain volume rendering*, as illustrated in the pipeline in Fig. 1.2. First, a 3D sample is digitized by a scanner, and then represented by scalar values on, e.g., a regular grid. Second, the data is decomposed or transformed by means of a mathematical framework into a compact data representation, which can be encoded and sent over the network in a lightweight form. When the data needs to be visualized, the inverse process is applied to reconstruct the original volume. This decomposition-reconstruction process is nowadays usually highly asymmetric [Fout and Ma, 2007]. That is, the data decomposition step is an offline process, which is not time critical, while the reconstruction process needs to be performed at run-time.

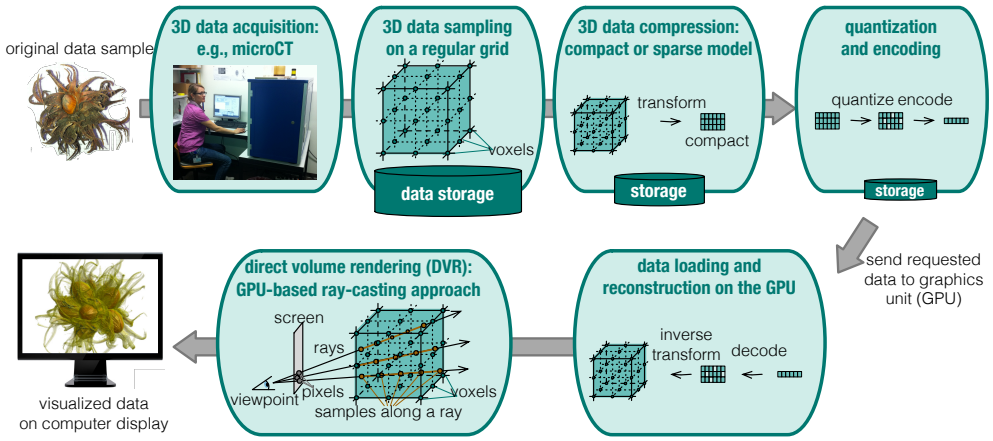


Figure 1.2: *Overview of the 3D direct volume rendering pipeline – starting from data acquisition, then applying data reduction, compact data representation, quantization, encoding and eventually resulting in a virtual image on a computer display.*

By definition, compact data representation can be *lossless* or *lossy*. Lossless algorithms ensure the maximum accuracy in the reconstruction and guarantee zero error. Lossy algorithms produce an *approximation* of the reconstructed volume up to a tolerated approximation error. In application domains where the data only has to look similar to the original, e.g., in direct volume rendering, lossy compression schemes are preferable since they provide a greater compression ratio. Note that even though some of the models are by definition lossless, they are in practice often utilized in their lossy form.

Another key point of large data visualization is to define a suitable multiresolution data model that can be traversed at run-time to adaptively select and load data portions at a certain resolution [Westermann, 1994; Cignoni et al., 1997; La Mar et al., 1999; Boada et al., 2001; Gobbetti et al., 2008; Crassin et al., 2009]. To do so, the original data is down-sampled to lower resolutions and made available in a hierarchical multiresolution data structure, where one hierarchy level corresponds to one data resolution. From this generated multiresolution hierarchy the visualized object is glued together out of different data resolutions - dependent on the viewpoint (see Fig. 1.3). Objects farther aways are typically loaded at lower resolutions, while details are added for objects closer to the viewpoint. Since the multiresolution data structure is typically organized as small subvolumes, this strategy makes it possible to load small data portions from large datasets out-of-core on demand. To reduce the cost of slow disk accesses for large volume data, caching or pre-fetching and bricking techniques are used [Guthe et al., 2002; Ljung et al., 2006b; Crassin et al., 2009].

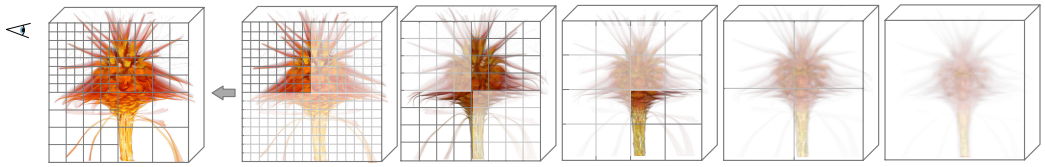


Figure 1.3: *Illustration of the idea of multiresolution data modeling: Subdivision of the dataset into equally-sized data blocks down-sampled to lower resolutions.*

The adaptive loading of data resolutions corresponds to offering different levels-of-detail (LOD), where the LOD is usually selected based on certain visual quality metrics or a rendering budget given for every displayed frame. In the context of out-of-core multiresolution volume representations, large volume datasets are also handled by using adaptive texturing schemes to compress and fit entire datasets into the GPU memory [Vollrath et al., 2006], or by using flat [Ljung et al., 2006b] multiresolution structures. Moreover, the rendering efficiency, which is important for the interaction with large volume datasets, can be improved using graphics

hardware acceleration to some extent, e.g., using texture-based hardware [Engel et al., 2001; Schneider and Westermann, 2003; Li et al., 2003].

To sum up: Data reduction in conjunction with large data visualization is of great importance, first, to save storage space at all stages of the processing and rendering pipelines, and second, to reduce time and cost of transmission during rendering and between the layers of the memory hierarchy. These compact data representations and their applicability in DVR are further elaborated in the next section.

1.2 Compact Data Representations

The general idea of a compact data representation is to express a dataset by a set of bases, which are used to reconstruct the dataset to its approximation when needed (see Fig. 1.4). Precisely speaking, a set of bases usually consists of the actual bases and coefficients describing the relationship between the original data and the actual bases. Typically, such basis sets constitute less data than the original dataset, capture the most significant features, and, moreover, describe the data in a format that is convenient/appropriate for adaptive data loading.

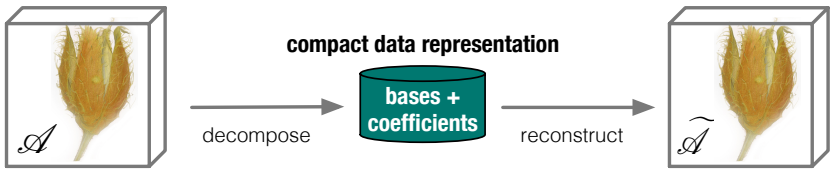
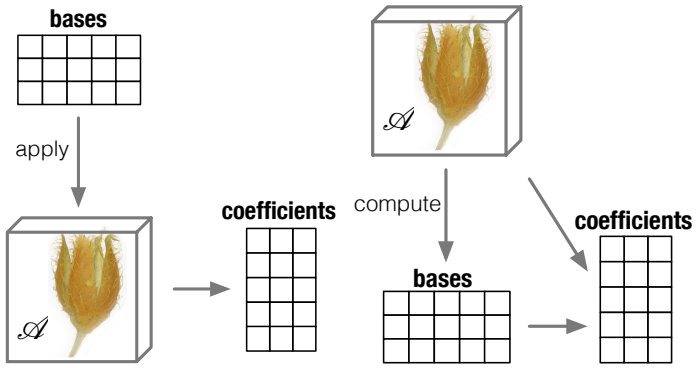


Figure 1.4: *Compact data representation for a volumetric dataset $\mathscr{A}$ by bases and coefficients that can be used to reconstruct the data to its approximation $\widetilde{\mathscr{A}}$ at run-time.*

Bases for compact data representation can be classified into two different types: *pre-defined* and *learned* bases. Pre-defined bases comprise a given function or filter, which is applied to the dataset without any a priori knowledge of the correlations in the dataset. In contrast, learned bases are generated from the dataset itself. Established examples of pre-defined bases are the Fourier transform (FT) and the Wavelet transform (WT). Well-known examples of learned bases are the PCA or the SVD. The main differences between pre-defined and learned bases are analyzed in the next section, while the applicability of compact representation by bases for interactive volume visualization is analyzed afterwards.

1.2.1 Pre-defined vs. Learned Bases

The compact model decomposition is often produced by the help of mathematical frameworks that represent the data by pre-defined or learned bases (see Fig. 1.5). For both basis types coefficients are produced in order to capture the relationship between the original data and the bases. While pre-defined bases are given a priori, learned bases are computed from the data itself. Using pre-defined or learned bases should be seen as two alternatives with both assets and drawbacks. In the following, the two basis types are analyzed according to different volume visualization-related criteria: the computational costs, the basis fit, the compactness, the feature-capturing ability, the axis-alignment, the multiresolution feasibility, and the reconstruction performance.



(a) Pre-defined bases (b) Learned bases

Figure 1.5: *Pre-defined vs. learned bases.*

Computational costs Methods using pre-defined bases are often computationally cheaper, while methods using learned bases require more precomputing time (to generate the bases). However, due to the given asymmetric computing power distribution during volume compression and decompression, the drawback of computationally more expensive approaches becomes less significant.

Basis fit, compactness and features capturing ability Pre-defined bases have to be selected according to the properties the available bases offer. For example, a Fourier transform (FT) is known to capture periodicity. Wavelets are available at a variety of simple as well as complex bases. However, more complex wavelets typically increase the encoding and decoding in terms of time and complexity. In contrast, learned bases are specifically generated for each dataset and therefore should be able to remove more redundancy from a dataset and be more compact.

Axis-alignment Pre-defined bases are applied by convolving the bases along each spatial axis, which corresponds to filtering with a given function kernel. For instance, wavelets (pre-defined bases) correspond to spatial averaging along the spatial axes. This makes it difficult to compactly represent unaligned 3D features. There has been much work on developing more powerfully oriented wavelet bases for multi-dimensional spaces [Do and Vetterli, 2001; Do and Vetterli, 2005]. However, such bases are still data-independent prescribed filters, and the compression efficiency gained over axis-aligned bases is limited [Wu et al., 2008]. Learned bases have an advantage over pre-defined ones since they search instead for correlations within the dataset and within all data directions. PCA (learned bases) for instance, represents the data in a more suitable coordinate system.

Multiresolution feasibility For multiresolution rendering, bases should support hierarchical data decompositions. For example, wavelets (pre-defined bases) are especially convenient for multiresolution volume rendering since they define a multiresolution hierarchy of coefficients, where each coefficient improves the approximation – higher-level coefficients are more important; small coefficients may be thresholded. For learned bases the hierarchy is typically constructed, e.g., with hierarchical vector quantization [Schneider and Westermann, 2003] or with residual-based data decompositions [Wu et al., 2008].

Reconstruction performance PCA-like approaches (learned bases) have the advantage that the components (or ranks) of the bases are ordered by the amount of data they describe and, hence, can be conveniently reduced without destroying cache coherence and without data repacking. Residual-based approaches and wavelets make collection of data more tedious since it is necessary to jump from one memory address to another to fetch the corresponding data portions.

Conclusion Using pre-defined bases is often computationally cheaper, while using learned bases requires more computing time (to generate the bases), but potentially removes more redundancy from a dataset. Volume rendering approaches based on pre-defined bases like wavelets are well studied, while there are fewer works on learned bases. Therefore, we would like to have a fresh look at the problem by learning preferential bases directly from the data, thus removing a bias in the approximation process. So far, this has been done in DVR using learned codebooks combined with vector quantization [Schneider and Westermann, 2003; Fout and Ma, 2007; Parys and Knittel, 2009], which require, however, large dictionaries if low distortion is desired. Moreover, while computational aspects and rendering performance of compact data representations have been extensively studied, these models are only beginning to be used for feature extraction.

1.2.2 Compression-domain Direct Volume Rendering

Compression-domain direct volume rendering concerns DVR approaches based on compact data representation and compression. As learned from the previous section, there are two main groups of compact data representations: Approaches using pre-defined bases and approaches using learned bases. Both approaches have been applied to compression-domain DVR.

Examples of pre-defined bases used in compression-domain direct volume rendering are the *discrete Fourier transform* [Dunne et al., 1990; Malzbender and Kitson, 1991; Levoy, 1992; Malzbender, 1993; Totsuka and Levoy, 1993; Chiueh et al., 1997] or the *discrete cosine transform* (DCT) [Yeo and Liu, 1995] (both frequency domain transforms) as well as the *discrete wavelet transform* (frequency domain transform while keeping spatial information), initiated for DVR by Muraki [Muraki, 1993] and refined in many works [Westermann, 1994; Grosso et al., 1996; Gross et al., 1997; Lippert et al., 1997; Ihm and Park, 1999; Kim and Shin, 1999; Rodler, 1999; Kim and Shin, 1999; Nguyen and Saupe, 2001; Guthe et al., 2002; Ljung et al., 2004; Wetekam et al., 2005; Wu and Qiu, 2005; Shen, 2006; Ko et al., 2008; Wang and Ma, 2008; Meftah and Antonini, 2009].

Examples of learned bases are *dictionaries*, which represent the entire datasets with a small set of pre-defined and learned codewords. Dictionaries used in DVR are *vector quantization* [Ning and Hesselink, 1992; Ning and Hesselink, 1993; Schneider and Westermann, 2003; Fout et al., 2005; Fout and Ma, 2007; Parys and Knittel, 2009] and *sparse coding* [Gobbetti et al., 2012].

Many compression-domain volume rendering systems have been developed. However, research in recent years has shown that only methods working directly on the graphics unit are competitive [Schneider and Westermann, 2003; Fout and Ma, 2007; Gobbetti et al., 2008; Yela et al., 2008; Nagayasu et al., 2008; Parys and Knittel, 2009; Mensmann et al., 2010; Iglesias Guitián et al., 2010; Gobbetti et al., 2012]. In particular, wavelet transform and vector quantization, often combined together, are standard tools for compression domain volume rendering. Wavelets are especially convenient for compressed DVR since they define a multiresolution hierarchy of coefficients [Westermann, 1994]. Similarly, there are hierarchical vector quantizers [Schneider and Westermann, 2003].

Compression-domain DVR becomes even more essential when dealing with time-varying volumetric datasets (see, e.g., [Weiss and Floriani, 2008]). Several strategies exist for this. First, the compact data representation methods can be extended in the dimensionality from 3D to 4D, which corresponds to exploiting the correlation of voxels in subsequent frames [Ibarria et al., 2003; Woodring et al., 2003; Wang et al., 2005b]. These approaches obtain a good compression ratio, but they need to have a small set of frames in memory for rendering a single time step. Second, the fourth dimension can be treated as in video encoding, i.e., by

exploiting temporal coherence [Westermann, 1995; Ma and Shen, 2000; Wang et al., 2005a; Shen, 2006; Ko et al., 2008; Mensmann et al., 2010; Wang et al., 2010; She et al., 2011; Jang et al., 2012]. That is, voxel data is delta-encoded with respect to some reference time step(s), which must be already available to decode the current one. Third, each time step can be encoded separately with a static volumes compression method [Gobbetti et al., 2012]. The main advantages of this method are its implementation simplicity and the full temporal random access; however, lower compression ratios are achieved without exploiting temporal coherence. Finally, hybrid forms of the above mentioned approaches can be applied [Lum et al., 2001; Binotto et al., 2003; Fout et al., 2005; Nagayasu et al., 2006; Wang et al., 2008; Cao et al., 2011].

For compact data representation, data reduction and compression are usually combined with data quantization and encoding. During the quantization step, floating point values are converted to integer precision or insignificant coefficients/codewords are truncated. For example, dictionaries are often designed such that they explicitly include a quantization step. The reduced and limited number of coefficients can be further compressed by additional encoding. Almost always, this is a lossless process. Well-known encoders are run-length encoders (RLE) or so-called entropy encoders like Huffman coding and arithmetic coding.

In compressed DVR, RLE has been widely used since the early applications of volume rendering [Avila et al., 1992; Lacroute and Levoy, 1994; Yeo and Liu, 1995; Grosso et al., 1996; Kim and Shin, 1999; Komma et al., 2007; Wang et al., 2010]. Entropy encoders, too, are frequently used in compressed volume rendering: For instance, Huffman encoding [Fowler and Yagel, 1994; Yeo and Liu, 1995; Rodler, 1999; Guthe et al., 2002; Ljung et al., 2004; Wetekam et al., 2005; Komma et al., 2007] or arithmetic encoding [Guthe et al., 2002; Xiong et al., 2003; Komma et al., 2007] has also been combined with RLE [Guthe et al., 2002; Schelkens et al., 2003]. In order to accelerate the decoding of variable length codewords in entropy encoding, *fixed length Huffman coders* were explicitly used and combined with RLE [Guthe et al., 2002; Shen, 2006; Ko et al., 2008]. Other approaches (mainly for wavelet coefficient encoding) [Lippert et al., 1997; Rodler, 1999; Ihm and Park, 1999] tried to fully avoid entropy encoding and RLE by using *significance maps* of wavelet coefficients and bit-wise encodings. Finally, there are models that have a particular hardware support during the decoding, for instance *block truncation coding* [Brown, 2001; Craighead, 2004; Ström and Pettersson, 2007; Agus et al., 2010; Nystad et al., 2012].

Compression-domain DVR is an active field of research. Nevertheless, until now, PCA-like methods have not yet been exploited for compression domain volume rendering. One reason for this might be that the extension of the PCA/SVD to higher-orders is not trivial. However, there have been achievements that describe how to use PCA-like methods for higher-orders, e.g., for volumes.

1.3 Higher-order Data Decompositions

The most common tools for data approximation with learned bases are the matrix SVD and the PCA. Their higher-order extensions are summarized under the term *tensor approximation* (TA). The first occurrence of TA was in [Hitchcock, 1927a; Hitchcock, 1927b]. The idea of multi-way analysis, however, is generally attributed to Catell in 1944 [Cattell, 1944; Cattell, 1952]. It took a few decades until tensor approximations received attention, which was by several authors in the field of psychometrics [Tucker, 1963; Tucker, 1964; Tucker, 1966; Caroll and Chang, 1970; Harshman, 1970].

The matrix SVD/PCA work on 2D matrix data and exploit the fact that the dataset can be represented with a few highly significant coefficients and corresponding reconstruction vectors based on the matrix rank reduction concept. The SVD and the PCA, being multilinear algebra tools compute (a) a rank-R decomposition, and (b) orthonormal row and column vector matrices. The extension to higher-orders is not unique and the two properties from the SVD are captured by two different models that are both given the term tensor approximation: the Tucker model [Tucker, 1963; Tucker, 1964; Tucker, 1966; Ten Berge et al., 1987; De Lathauwer et al., 2000a; De Lathauwer et al., 2000b; Kolda and Bader, 2009] preserves the orthonormal factor matrices while the CP model (from CANDECOMP [Caroll and Chang, 1970] and PARAFAC [Harshman, 1970]) preserves the rank-R decomposition.

Generally speaking, a *tensor* is a term for a higher-order generalization of a vector or a multidimensional array; for example, a scalar is a 0^{th}-order tensor, a vector is a 1^{st}-order tensor and a matrix is a 2^{nd}-order tensor. Tensors with more than three modes (data directions) are called higher-order tensors. In TA approaches, a multi-dimensional input dataset in array form, i.e., a tensor, is factorized into a sum of rank-one tensors or into a product of a core tensor (coefficients that describe the relationship to input data) and matrices (bases), i.e., one for each dimension. This factorization process is generally known as *tensor decomposition*, while the reverse process of the decomposition is the *tensor reconstruction*.

Tensor decompositions have been widely studied in other fields and were reviewed [Moravitz Martin, 2004; Kolda and Bader, 2009; De Lathauwer, 2009] and summarized [Smilde et al., 2004; Kroonenberg, 2008]. Since TA was emerging from different disciplines, it was developed under various names. In particular, the Tucker model is known in the literature under multiple terms. The CP model was independently developed under the terms CANDECOMP and PARAFAC, therefore it is sometimes referenced with a single name. The Tucker model takes its name from Tucker, who initially worked on the *three-mode factor analysis* (3MFA), which is sometimes referred to as the Tucker3 model. [Kroonenberg and De Leeuw, 1980; Ten Berge et al., 1987; Kroonenberg, 2008] called it the

three mode PCA (3MPCA). Similarly the model was referenced as *N-mode PCA* by [Kapteyn et al., 1986]. [De Lathauwer et al., 2000a] captured all these previous works and wrote down the generalization of the SVD as *multilinear singular SVD*, which is usually termed as higher-order SVD or HOSVD. Thereafter, [Vasilescu and Terzopoulos, 2002; Vasilescu and Terzopoulos, 2004] called it *N-mode SVD*. In this thesis, we either refer to the Tucker model to refer to the tensor decomposition, or we refer to HOSVD in order to describe the underlying procedure to produce the higher-order tensor decompositions.

Generally, PCA-like methods are able to extract the main data direction of the dataset and represent the data in a different coordinate system/subspace such that it makes it easier for the user to find the major changes or variance within the dataset. To exploit this feature, we aim to use its higher-order extensions, namely TA, within volume rendering in order to extract features and to reduce the actual amount of coefficients needed for storage. In that way, features occurring at multiple scales should be found with the help of the rank-R decomposition. When using TA for compression domain volume rendering, we need algorithms that decompose and reconstruct the dataset. There are a few commonly available TA implementations, e.g, several MATLAB toolboxes are available. The *MATLAB tensor toolbox* [Bader and Kolda, 2006; Bader et al., 2012] is the most comprehensive one; other toolboxes are the *N-way toolbox* [Andersson and Bro, 2000], the *PLS toolbox* [Wise and Gallagher, 2007], and *CUBatch* [Gournévec et al., 2005] – a MATLAB interface. The *Multilinear Engine* [Paatero, 1999] is a FORTRAN-based library, which mainly supports CP and PARAFAC2. Just recently, the Tensorlab toolbox for MATLAB [Sorber et al., 2013] was released, which provides many tensor decomposition algorithms, optimization procedures and tensor visualizations; however, it was released after this thesis submission and could therefore not be considered. For direct volume visualization, an efficient memory handling is crucial. In particular for large datasets, we found MATLAB not to be applicable. There are high-performance tensor libraries in C++ [Landry, 2003; Garcia et al., 2005; Zass, 2006]. However, they only provide basic tensor operations and no tensor decomposition and reconstruction algorithms.

Tensor approximation has been used in many areas among which there are applications in the domain of visualization and computer graphics. An overview of theses is given in the next section.

1.3.1 Applications in Visualization and Graphics

TA approaches have been applied to a wide range of application domains. Starting from psychometrics, in recent years, tensor approximation has been applied to visual data. A highly studied area is TA used for image ensembles (see [Shashua and Levin, 2001; Vasilescu and Terzopoulos, 2002; Wang and Ahuja, 2004; He

et al., 2005; Shashua and Hazan, 2005; Wang and Ahuja, 2005; Wang and Ahuja, 2008; Yan et al., 2009; Morozov et al., 2011]) and/or TA used for pattern recognition, e.g., [Shashua and Levin, 2001; Wang and Ahuja, 2005; Savas and Eldén, 2007; Schultz and Seidel, 2008; Ergin et al., 2011; Liu et al., 2012]. In (real-time) rendering, tensor decompositions have recently been used as method for global illumination models, e.g., for bidirectional reflectance distribution functions (BRDFs) [Sun et al., 2007; Bilgili et al., 2011] or precomputed radiance transfer (PRT) [Tsai and Shih, 2006; Sun et al., 2007; Tsai and Shih, 2012]. TA, furthermore, is successfully used for bidirectional texture functions (BTFs) [Furukawa et al., 2002; Vasilescu and Terzopoulos, 2004; Wang et al., 2005b; Wu et al., 2008; Ruiters and Klein, 2009; Ruiters et al., 2012; Tsai and Shih, 2012], texture synthesis [Wu et al., 2008], time-varying visual data, e.g., [Wang et al., 2005b; Wu et al., 2008], 3D face scanning [Vlasic et al., 2005] and animation (see [Vasilescu, 2002; Mukai and Kuriyama, 2007; Perera et al., 2007; Wampler et al., 2007; Krüger et al., 2008; Min et al., 2010; Liu et al., 2011]). So far, TA has not yet been used to model datasets for direct volume rendering (DVR).

Key challenges for DVR of large datasets are that a hierarchical data structure can be used and that the reconstruction of TA is possible in real-time. For instance, for datasets larger than the main memory, out-of-core approaches as proposed for TA in [Wang et al., 2005b] are needed. A complete hierarchical multiscale TA system has been proposed in [Wu et al., 2008]. Their data hierarchy consists of tensor decomposed subvolumes with each level having a progressively decreasing rank-reduction level. All subvolumes or (volume) bricks – as they are called – on one hierarchy level, representing the residual to the parent level, are treated as a tensor ensemble. As a result, each hierarchy level consists of one single rank-reduced core tensor and multiple factor matrices. However, for interactive reconstruction and visualization, many temporary results must be cached in the hierarchy at run-time. Instead, we built a multiresolution TA octree where each lower resolution brick is an autonomous tensor decomposition that can be independently reconstructed and rendered on demand, attaining interactive speed.

Furthermore, for compact data representation, the encoding of numerical values is fundamental and hence an appropriate quantization must be devised. We refrain from variable-length coding at this point to avoid the corresponding costly decompression. Fixed linear quantization for the factor matrices (8-bit) and core tensor (8-20-bit) has been proposed in [Wu et al., 2008]. We investigated more tensor-specific linear and non-linear quantizations, suitable for fast reconstruction implementation on the GPU. In particular, the distribution of the core tensor coefficients can benefit from logarithmic quantization.

1.4 Summary

1.4.1 Hypothesis: Unified Framework

The challenge of this dissertation is the interactive visualization of large volume datasets, which is illustrated in Fig. 1.6. We have acquired large 3D datasets, which should be interactively visualized in order to allow an explorative analysis by an expert in the field. With the recent improvements in data acquisition tools, nowadays, huge amounts of datasets are acquired. Typically, the sizes of these datasets exceed the available memory and the computing requirements exceed today's graphics hardware. In order to visualize large datasets, we therefore need tools that can be applied prior to the actual data visualization. These very large data volumes not only represent an ever-growing amount of information, but also exhibit an increasing level of structural complexity in space and time, resulting in a high degree of complexity at different scales. A common approach is to find a single tool that performs data reduction and feature extraction in one, meaning that we remove the irrelevant data parts while we keep the essential features and information within the dataset. The reduced dataset is usually stored within a compact data representation, which makes it possible to experiment with a few parameters in order to adjust the level of data reduction and the scale or level of feature extraction.

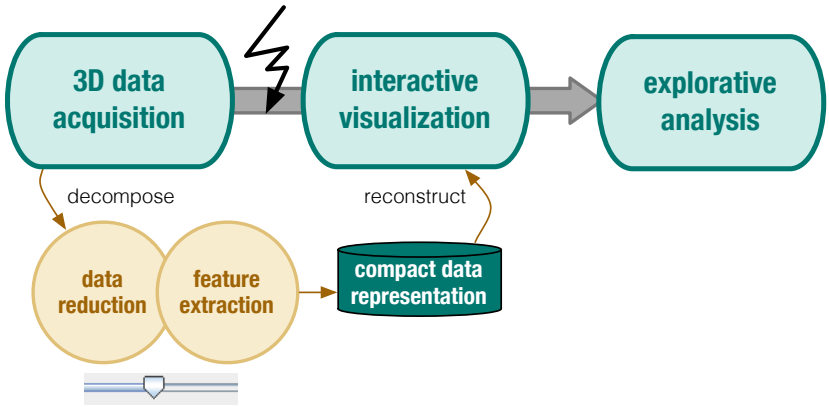


Figure 1.6: *Interactive visualization of large 3D datasets and where to hook in to apply data reduction and feature extraction.*

The main goal followed in this thesis is to find one unified framework for large volume visualization, which does three tasks in one: (a) reduces the actual amount of data, (b) extracts relevant features from the dataset, and (c) visualizes the data directly from the framework's coefficients. The idea is to have as few parameters as possible to steer and tune the aforementioned tasks. In this thesis, higher-order

tensor approximation was chosen as a unique framework since it provides learned bases and has one single parameter (the tensor rank), which enables data reduction and feature extraction; moreover, it offers properties that fit multiresolution volume rendering approaches. The TA framework is tested on several aspects matching large data visualization.

1.4.2 Contributions

The thesis is structured into a classical IMRAD organization (introduction – materials and methods – results and discussion). For materials and methods, large volume datasets such as those used in this thesis as well as the visualization methods and concepts as applied to the given datasets are presented and summarized. Then results are given in different chapters, with each chapter addressing a different key issue in the interactive visualization pipeline. To wrap up, the results are discussed and future research directions are outlined. The specific contributions are highlighted next.

Materials During this thesis, a wide range of test datasets were acquired. Most of these datasets were acquired with micro-computed tomography (μCT). The test datasets represent different sizes (256^3, 512^3, 1024^3, 2048^3 and larger), which were useful to test the algorithms and to discover bottlenecks and to set benchmarks for the visualization performance. The datasets are offered to the public and made available to other researchers. This is an important contribution since it is often difficult for computer graphics researchers to get real large test datasets. Furthermore, it is critical to test algorithm performance under similar conditions to previous contributions to allow more effective comparison. Additionally to those test datasets, even larger datasets from fossil teeth were acquired with phase-contrast synchrotron tomography, which makes it possible to scan tissue at the micrometer scale. These datasets exhibit complex 3D internal structures. With the help of those datasets, we show how features at multiple scales can be visualized by the help of TA. The dataset sizes go up to 32GB.

Methods The method of choice for large volume visualization was out-of-core multiresolution direct volume rendering (DVR), specifically GPU-based ray casting, which makes it possible to obtain volumetric sections and transparent regions within a dataset. DVR is well studied in combination with out-of-core multiresolution approaches. The underlying data structures are hierarchical and allow effective data reduction. We chose tensor approximation as a unified framework to compactly represent datasets and show how TA features and TA properties can be exploited to effectively and efficiently map to multiresolution volume rendering needs.

Results The introduced TA methods and concepts are tested according to their *interactivity*, *multiresolution-feasibility*, and *multiscalability*. The interactivity and the multiresolution-feasibility are analyzed by means of two different multiresolution DVR systems, which were implemented during this thesis: (1) a simple system, where every data brick (data portion) is represented with its own small tensor decomposition (per brick matrices and core tensor), and (2) a more complex system, where every data brick is stored as a tensor decomposition from global bases (per brick core). The latter approach should, in principle, support a multiscale, multiresolution TA hierarchy to independently and separately control data reconstruction at different scales (of patterns and feature sizes) as well as of the spatial reconstruction resolution.

The thesis includes an analysis of the data reduction achieved by the tensor-specific quantization scheme and the asymmetric interactive performance achieved (offline pre-processing and real-time rendering). Even though we aim for offline preprocessing routines, during this thesis, we show improvements for a faster HOSVD decomposition and, in particular, we show that it is feasible to decompose a large tensor decomposition from a 32GB sized volume dataset in order to produce the global bases. The multiscalability is verified with various examples of visualizations and error metrics. In particular, it is shown how TA parameters can be used to steer multiscale features and how TA compares to other state-of-the-art multiscale approaches. The implementation of the proposed TA algorithms is mostly contributed to an open-source library (vmmlib), which allows other researchers to make comparisons with our methods more easily. Moreover, a real-time GPU-based tensor reconstruction, which exploits parallel computation, was introduced for the first time. Finally, for data reduction, we introduce a tensor-specific quantization scheme.

Discussion The chosen multiresolution and multiscale TA visualization approach makes it possible to focus, from the beginning of the rendering pipeline, on those features in the dataset that are relevant to the user, thus considerably enhancing rendering efficiency and permitting interactive exploration of large volume datasets. At the same time, explorative flexibility is guaranteed through a multiscale approach, which permits inspection of features at user-defined levels of scale. This thesis introduces a method for using TA in a volume visualization system for large datasets and at the same time it opens many new research directions, which can be followed as future work. Some of these future research directions are outlined after the discussion of the new achievements.

CHAPTER 2

MATERIALS

2.1 Non-invasive Tissue Analysis

In this thesis, several datasets were used to verify the developed algorithms. The actual datasets are summarized in App. A. In this chapter, we focus on the importance of the interactive 3D visualization of tissues relevant in biomedical applications. The scanning of real tissues allows researchers from various fields to conduct the virtual analysis of their samples on a computer screen. The advantage of this procedure is that a researcher can look into such samples without cutting them. Typical scanning approaches are based on X-ray tomography, which restricts the analysis for living samples to some extent. For the verification of the developed algorithms in this thesis, only dead samples and various X-ray tomography approaches were used.

Non-invasive analysis of organic structures, tissue and materials with microtomographic techniques has seen rapid development over the past few years. Microcomputed X-ray tomography (μCT) has now become a standard tool, e.g., in biomedical research. μCT is an X-ray technique that produces 3D images of tissue with voxel sizes down to approximately 1μ or even smaller. Compared to conventional CT, μCT uses a smaller field of view with a high resolution detector. During scanning, the X-ray attenuation of the material properties is captured in order to reconstruct the 3D structure. Thus, μCT can be used to study materials including bone, teeth, medical implants, snow, textiles, concrete and similar. As a relatively recent X-ray technology, *synchrotron tomography* (ST) has opened up new areas of research at the sub-micrometer level. In particular, *phase contrast synchrotron tomography* (pcST) has become of special interest for the growth structures analysis in hard tissues of living and fossil species [Tafforeau and Smith, 2008; Friis et al., 2007] since it offers a high contrast of structures in the sample.

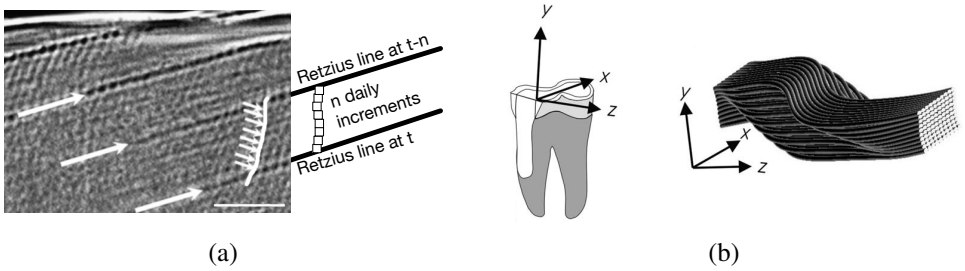
The grand challenge today is thus to make the *implicit* information contained in structural volume data *explicitly* available. Since many internal structures are in the micrometer domain, the data size of one volume block typically exceeds the limits for interactive visualization on modern graphic systems. Therefore, preprocessing of the datasets prior to rendering for visual exploration is needed. Meeting this challenge requires not only solutions for large-scale volume rendering, but more specifically a method to reduce the dataset size and a method to represent volume features at their feature scale.

An example case study in this thesis is taken from the application of virtual analysis of fossil tissue in paleoanthropology. For example, patterns of daily enamel deposition in fossil hominid teeth are imaged with pcST and counted to estimate the age at death of a fossil specimen. However, standard approaches are restricted to the analysis of serial 2D cross-sections through data volumes, while the actual growth microstructures have complex 3D shapes [Jiang et al., 2003; Macho et al., 2003]. Nowadays, X-ray scanning with pcST is possible, however, the

resulting dataset sizes are large, which limits the interactive exploration of such datasets. Hence there is a need for tools to visualize 3D microstructural features interactively, which is the topic of this thesis. The application of virtual analysis of patterns in teeth was chosen as an example for the verification of the developed system and algorithms and is described next.

2.2 Structures and Periodicity in Dental Tissue

The dental enamel dataset is interesting since it represents periodic growth structures that occur at different levels of scale, and exhibit different spatial orientations. Human tooth enamel has a microstructure that is roughly comparable to a bunch of densely packed fibers (so-called prisms). During dental enamel formation, each dental enamel prism elongates in a centrifugal direction through the daily apposition of a small segment of enamel. Daily growth increments are visible as surfaces perpendicular to the longitudinal direction of the prisms. In addition, approximately weekly growth halts are visible as so-called Retzius lines (see Fig. 2.1(a)). This data characteristic is a well-suited test case for the multiscale feature visualization that is the aim of this thesis. As can be seen in Fig. 2.1(b), the dental growth patterns exhibit 3D periodic [Macho et al., 2003], which need a 3D explorative visualization system rather than only a 2D analysis as performed with physical cross-sections. Furthermore, the spatial scale and orientation of these structures is highly characteristic for each feature.



(a) (b)

Figure 2.1: *(a) Cross-sectional image of tooth enamel (pcST scan; scale = 50 microns) [Tafforeau and Smith, 2008]. Small arrows: cross striations; large arrows: Retzius lines. The direction of the growth prisms is orthogonal to the cross striations. (b) A simplified model of the dental growth structures (prisms) after Macho et al. [Macho et al., 2003].*

In the multiscale chapter (Chap. 6), we first used synthetic volumes simulating dental growth structures, which we generated after Macho et al. [Macho et al., 2003] as in Fig. 2.1(b). To complement this, we used pcST volumes of great ape teeth. The pcST were available from experiment number 20080205 at the Swiss Light Source.

CHAPTER 3

METHODS

3.1 Direct Volume Rendering (DVR)

Direct volume rendering (DVR) is the visualization technique considered in order to show the full volumetric data (rather than only a surface as usually visualized by mesh approaches). DVR has the advantage that it cuts through the volume and transparent volume areas can be displayed. This is important, for instance, to display different tissues. A 3D volume on a uniform grid (to which we restrict ourselves), is typically organized into equally sized volume elements (voxels), which cover a certain volume area and which are represented by a scalar value, as illustrated in Fig. 3.1.

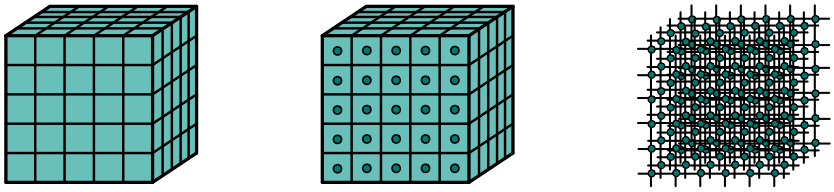


Figure 3.1: *A DVR representation of a volume: (a) discrete grid for volume dataset, (b) voxel centers represented by scalar values, (c) voxels on uniform grid.*

In other words, the visualization technique considered in this thesis is mainly the direct display of scalar fields defined over a three-dimensional space. A scalar field $f : \mathbb{R}^3 \to \mathbb{R}$ assigns a value to each point in the volume and is typically sampled by $f_{ijk} \in \mathbb{R}$ on a grid of voxels $P_{ijk} \in \mathbb{R}^3$. A continuous function is defined by interpolating between the corners of the cell (ijk) containing P (Fig. 3.2) by $f(P) = \sum_{ijk}^{i+1\,j+1k+1} \psi_{ijk}(P) \cdot f_{ijk}$, often using tri-linear interpolation functions ψ_{ijk}.

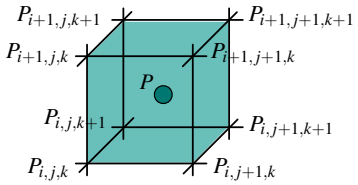
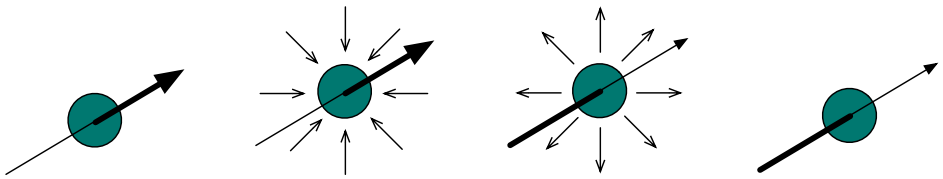


Figure 3.2: *Volume cell containing point P.*

3.1.1 Volume Rendering Integral

Most direct volume visualization approaches use the so-called *volume rendering integral* for visualization, proposed in [Kajiya and Herzen, 1984] and well

reviewed in [Moreland, 2004], which was further formally developed with the emission and absorption theorem ([Max, 1995]) and relies on geometric optics. According to geometric optics, light is assumed to propagate along straight lines unless interaction between light and a participating medium takes place [Engel et al., 2006]. The following types of interactions are usually considered (Fig. 3.3): *emission*, medium increases radiative energy; *absorption*, medium absorbs radiative energy; and *scattering*, medium scatters and changes direction of light. Emission, absorption, and scattering affect the amount of radiance I (light energy) along a light ray and ergo describes the light transfer along one direction of a light ray.



(a) emission (b) in-scattering (c) out-scattering (d) absorption

Figure 3.3: *Light interactions along one ray and through one voxel [Engel et al., 2006].*

Since the full computation of the light energy is computationally expensive, simplified versions are often used. The *emission-absorption* model, for example, is one of the most widely used models for volume rendering. The emission-absorption model leads to the *volume rendering equation*, Eq. (3.1), for the radiance I for a given position s along one single ray, where κ is the absorption coefficient and q is the emission source term.

$$\frac{dI(s)}{ds} = -\kappa(s) \cdot I(s) + q(s) \tag{3.1}$$

The volume rendering equation can be solved by the so-called *volume rendering integral* (DVRI), Eq. (3.2), which solves Eq. (3.1) along the direction of light from a starting point $s_0 = 0$ to the endpoint $s_1 = D$.

$$I(D) = I_0 \cdot e^{-\int_0^D \kappa(t)dt} + \int_{s_0}^{D} q(s) \cdot e^{-\int_s^D \kappa(t)dt} ds \tag{3.2}$$

In other words, the volume rendering integral depicts the attenuation of the light emitted by an initial light source on its way through a medium to the viewpoint. The term $\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(t)dt$ defines the optical depth between two positions s_1 and s_2, which is a measure for the duration a light ray may travel before

it is absorbed. Small values of the optical depth correspond to transparent mediums, high values to opaque mediums. This definition of the transparency α as in Eq. (3.3) leads to the DVRI as in Eq. (3.4).

$$\alpha(s_1, s_2) = e^{-\tau(s_1,s_2)} = e^{-\int_{s_1}^{s_2} \kappa(t)dt} \tag{3.3}$$

$$I(D) = I_0 \cdot \alpha(s_0, D) + \int_{s_0}^{D} q(s) \cdot \alpha(s, D)ds \tag{3.4}$$

A lot of research effort is put into efficiently computing this volume rendering integral (see e.g., [Engel et al., 2006; Schlegel, 2012]). Typically, numerical approximations are used since the integral cannot be evaluated analytically. This can be done by splitting the integral into several integration segments from points s_{i-1} to points s_i. The transparency α_i and color contribution c_i of the i-th segment is formulated as in Eq. (3.5)–(3.6).

$$\alpha_i = \alpha_i(s_{i-1}, s_i) \tag{3.5}$$

$$c_i = \int_{s_{i-1}}^{s_i} q(s) \cdot \alpha(s, s_i)ds \tag{3.6}$$

This leads to the *discretization of the volume rendering integral*, usually formulated by a Riemann Sum as in Eq. (3.7)–(3.8).

$$I(D) \approx \sum_{i=0}^{n} c_i \cdot \prod_{j=i+1}^{n} \alpha_j \tag{3.7}$$

$$c_0 = I(s_o) \tag{3.8}$$

Incorporating alpha blending of multiple transparent of voxels, we get Eq. (3.9).

$$I(D) \approx \sum_{i=0}^{n} \alpha_i \cdot I_i \cdot \prod_{j=i+1}^{n} (1 - \alpha_j) \tag{3.9}$$

DVR approaches A number of methods have been developed to sample rays and compute the DVRI of Eq. (3.9), mostly based on 3D texturing, splatting or ray casting [Engel et al., 2006]. During the last decade, ray casting has become the major trend for volume rendering, since the development of programmable graphics hardware [Lindholm et al., 2001] has become powerful and permits

highly interactive frame rates nowadays. In fact, ray-based volume visualization approaches had already been introduced quite some time ago [Tuy and Tuy, 1984; Levoy, 1988]. However, despite their superior image quality, they only became attractive for interactive applications due to GPU computing approaches, which eventually satisfy the enormous processing demands of ray-based methods, e.g., [Kruger and Westermann, 2003; Roettger et al., 2003]. GPU-based ray casting is therefore the choice of rendering method in this thesis.

3.1.2 GPU-based Ray Casting

The concept of ray casting is illustrated in Fig. 3.4: We shoot rays from the viewpoint towards the volumetric dataset, trace those rays through the volume, and project the accumulated volume information onto a screen plane. There is one ray shot per screen pixel, which corresponds to a so-called *image space* algorithm, where the contribution of the objects is evaluated per pixel. To obtain the final pixel value, samples at a defined frequency are taken and evaluated according to the DVRI. The sample values are mapped to their final color and opacity according to a pre-defined look-up table, the so-called transfer function.

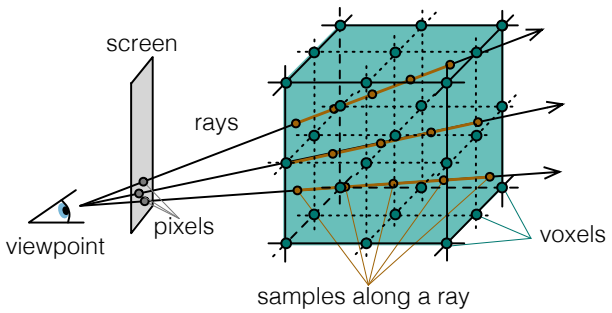


Figure 3.4: *Direct volume rendering with volume ray casting.*

Specifically, for GPU-based ray casting, the rays are sent through the volume by the CPU, then the GPU computes all the samples along the ray and renders the final pixel. For the interactive display of large volume data, ray casting is in practice implemented as out-of-core multiresolution direct volume rendering.

3.2 Out-of-core Multiresolution DVR

As learned from the previous section, the visualization of large datasets is an ongoing challenge. Further computing time can be saved by decreasing the actual amount of data sent to the volume renderer. The two key issues here are to define

(1) a suitable multiresolution model in order to generate a level-of-detail approximation hierarchy over the input volume, and (2) an out-of-core memory data management, since the full dataset typically does not fit the GPU memory.

The most straightforward way to deal with datasets larger than the available memory is the *divide-and-conquer* approach. Accordingly, we subdivide our input volume into equally sized subvolumes, so-called *bricks* (see Fig. 3.5. By subdividing or bricking the volume dataset into bricks, each data portion becomes small enough to be loaded on demand for visualization by the out-of-core data management. In multiresolution volume visualization one typically works with equally sized bricks, since it simplifies the fitting of the resolutions.

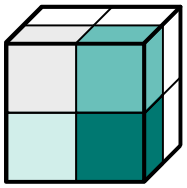


Figure 3.5: *Bricking: Subdivision of a volume into eight equally-sized subvolumes.*

Usually, there is a data management system that accesses, loads and stores the bricks/subvolumes. This is usually referred to as an out-of-core data management system, which organizes the data blocks. During rendering, the data blocks involved are copied onto the GPU and processed for visualization. This means that there is a copying of data from the different memory layers involved. Since some memory, such as the GPU memory, is limited, the data blocks are removed again once they are not used anymore (oldest out first). At the same time, loaded blocks are cached such that frequently accessed data blocks do not need to be uploaded multiple times. I/O transfer is still a limiting bottleneck. Therefore caching mechanisms are crucial. Out-of-core rendering extensions have, been presented by, for example, [Pascucci, 2000; Correa et al., 2002; Wang et al., 2005b; Crassin et al., 2009]. The out-of-core data management and caching goes along with the concept of multiresolution data modeling. Next, we take a closer look at the multiresolution model and its possible data structures.

3.2.1 Multiresolution Data Approximation

The basic idea of multiresolution volume visualization is to adjust the resolution of the rendered data according to the viewer's position – the viewpoint – or other rules/guidelines. Fig. 3.6 illustrates this concept in 3D and in 2D for simplification. The data close to the viewpoint is typically rendered at higher resolution, while data further away is rendered at lower resolutions. Correspondingly, the multiresolution composition greatly depends on the viewpoint.

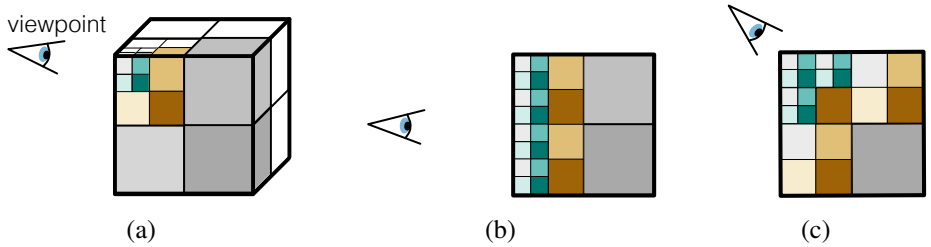


Figure 3.6: *View-dependent multiresolution representation of a volume. (a) High-resolution data is close to the viewpoint, while data regions further away are rendered at lower resolutions, (b-c) Two different viewpoints can create two different multiresolution representations, illustrated here in 2D.*

In order to have the dataset available at rendering time in the desired resolution, the full dataset is usually preprocessed and organized into a multiresolution data structure. In the literature, flat [Ljung et al., 2006b] or hierarchical [Gobbetti et al., 2008; Crassin et al., 2009; Iglesias Guitián et al., 2010] multiresolution structures in conjunction with adaptive loaders can be found. Here, we use a hierarchical level-of-detail based data structure referred to as an *octree*. An octree as visualized in Fig. 3.7 contains the data at the original resolution in the leaf nodes while all other octree nodes represent lower-resolution versions of the data. The octree used in this thesis is constructed bottom-up. A preprocessing method gradually averages/subsamples eight bricks into a parent node or brick of lower resolution until the root node is generated. The root node and all the other nodes are represented by equally sized bricks B^3. For generalized preprocessing, the input dataset is typically zero-padded to a power-of-two input dataset size. With an octree, the data can be traversed top-down from the coarsest resolution, the root, gradually refining data regions with higher resolutions.

During rendering, each block consists of the same number of data voxels, but spans a different actual spatial volume area. For the final visualization, the bricks need to be glued together, which opens the challenge of resolving brick border artifacts. Such artifacts are inherent to all brick-based lossy compression methods, and can be alleviated, at the cost of higher rendering time, by interblock interpolation through sampling neighboring bricks [Ljung et al., 2006a; Beyer et al., 2008] or by using deferred filtering [Fout et al., 2005; Fout and Ma, 2007; Gobbetti et al., 2012].

In order to further compress each data brick, a compact data representation can be applied for each brick. In the next section, the chosen compact data representation – tensor approximation – is described and defined in detail.

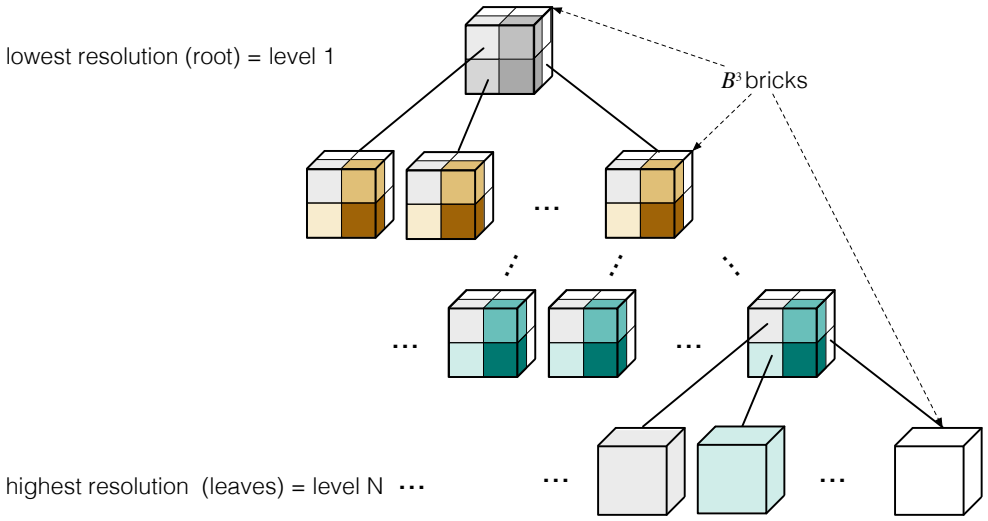


Figure 3.7: *Octree data structure used for multiresolution volume visualization. Eight children always form a subsampled/averaged parent node (bottom-up process).*

3.3 Tensor Approximation (TA)

For this thesis, tensor approximation (TA) was the chosen framework in order to perform data reduction and feature extraction for direct volume rendering. TA consists of two parts: The *tensor decomposition* transforms a multidimensional dataset into a compact data representation, while the *tensor reconstruction* approximates the original volume from the compact data representation. TA is a higher-order extension of the singular value decomposition (SVD) or the principal component analysis (PCA). The linear algebra background about SVD/PCA can be found in App. B. In this section, we explain the notation and definitions of tensor algebra as well as the models and algorithms used in TA.

3.3.1 Notation and Definitions

The notation taken here is inspired by that ones of De Lathauwer et al. [De Lathauwer et al., 2000a], Smilde et al. [Smilde et al., 2004], and Kolda and Bader [Kolda and Bader, 2009], who follow the notation proposed by Kiers [Kiers, 2000]. Other standards have been proposed as well (see [Harshman, 2001] and [Harshman and Hong, 2002]). To illustrate higher-order extensions we mostly make examples of order three. For details on the higher-order extensions of different products related to computing with tensors (see App. C).

General

A tensor is a multi-dimensional array (also called an N-way data array): a zeroth-order tensor (tensor0) is a scalar, a 1^{st}-order tensor (tensor1) is a vector, a 2^{nd}-order tensor (tensor2) is a matrix, and a 3^{rd}-order tensor (tensor3) is a volume. We consistently use the letter A to represent the data. This follows the notation of, e.g., [De Lathauwer et al., 2000a; De Lathauwer et al., 2000b; Wang et al., 2005b; Wu et al., 2008; Tsai and Shih, 2012][1]. We use lower case letters for a scalar a, lower case boldfaced letters for a vector $\boldsymbol{a}$ in $\mathbb{R}^{I_1}$, capital boldfaced letters for a matrix $\mathbf{A}$ in $\mathbb{R}^{I_1 \times I_2}$, and calligraphic letters for a 3^{rd}-order tensor $\mathscr{A}$ in $\mathbb{R}^{I_1 \times I_2 \times I_3}$ (see Fig. 3.8).

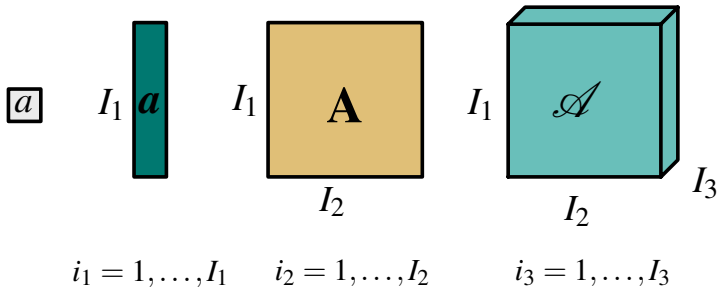


Figure 3.8: *A tensor is a multi-dimensional array: a zeroth-order tensor (tensor0) is a scalar a, a 1^{st}-order tensor (tensor1) is a vector* $\mathbf{a}$*, a 2^{nd}-order tensor (tensor2) is a matrix* $\mathbf{A}$*, and a 3^{rd}-order tensor (tensor3) is a volume* $\mathscr{A}$*.*

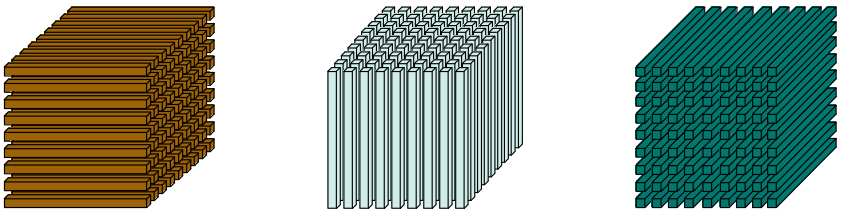
The *order* of a tensor is the number of data directions, also referred as *ways* or *modes*. Along a mode j, the index i_j runs from 1 to I_J. By using lower script indices for the modes, we can extend the index scheme to any order, i.e., $I_1, I_2, I_3, I_4, \ldots$. The i^{th} entry of a vector $\boldsymbol{a}$ is denoted by a_i, an element (i_1, i_2) of a matrix $\mathbf{A}$ is denoted by $a_{i_1 i_2}$, and an element (i_1, i_2, i_3) of a 3^{rd}-order tensor $\mathscr{A}$ is denoted by $a_{i_1 i_2 i_3}$.

The general term *fibers* is used as a generalization for vectors taken along different modes in a tensor (see Fig. 3.9). A fiber is defined by fixing every index but one. A matrix column is a mode-1 fiber and a matrix row is a mode-2 fiber. 3^{rd}-order tensors have column, row, and tube fibers, denoted by $\boldsymbol{a}_{i_1}, \boldsymbol{a}_{i_2}$, and $\boldsymbol{a}_{i_3}$, respectively. Sometimes, fibers are called mode-n vectors.

Slices are two-dimensional sections of a tensor (e.g., one fixed index in a tensor3). For a 3^{rd}-order tensor $\mathscr{A}$, there are, for example, frontal, horizontal, and lateral slices, denoted by $\mathbf{A}_{i_1}, \mathbf{A}_{i_2}$, and $\mathbf{A}_{i_3}$, respectively, (see Fig. 3.10).

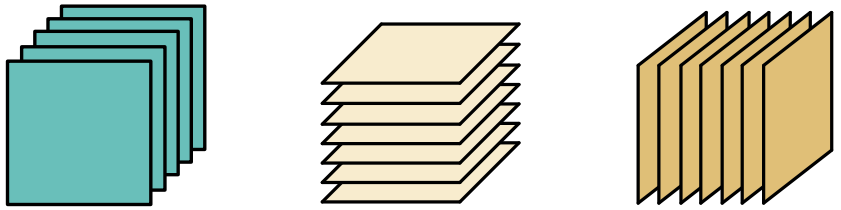
For computations, a tensor is typically reorganized into a matrix what we denote as *tensor unfolding* (sometimes called *matricization*). There are two main

[1] In other areas, however, as for example in statistics, it is common to use the letter X for the data [Kiers, 2000; Kolda and Bader, 2009].



(a) rows (b) columns (c) tubes

Figure 3.9: *Fibers of a tensor3 $\mathscr{A}$: (a) rows $\mathbf{a}_{i_1}$, (b) columns $\mathbf{a}_{i_2}$, and (c) tubes $\mathbf{a}_{i_3}$.*



(a) frontal slices $\mathbf{A}_{i_3}$ (b) horizontal slices $\mathbf{A}_{i_1}$ (c) lateral slices $\mathbf{A}_{i_2}$

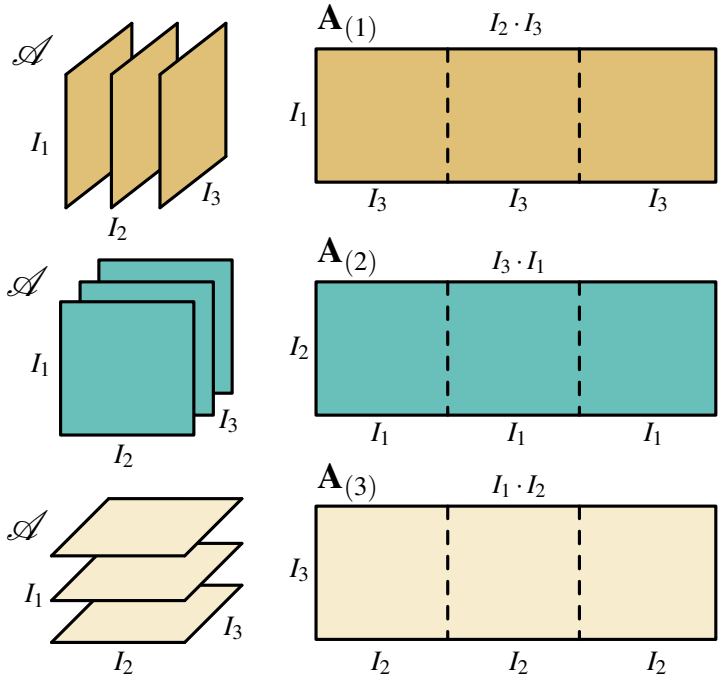
Figure 3.10: *Frontal, horizontal and lateral slices of a tensor3 $\mathscr{A}$.*

unfolding strategies, *backward cyclic unfolding* [De Lathauwer et al., 2000a] and *forward cyclic unfolding* [Kiers, 2000] (see Fig. 3.11). An unfolded tensor in matrix shape is denoted with a subscript in parentheses, e.g., $\mathbf{A}_{(n)}$.

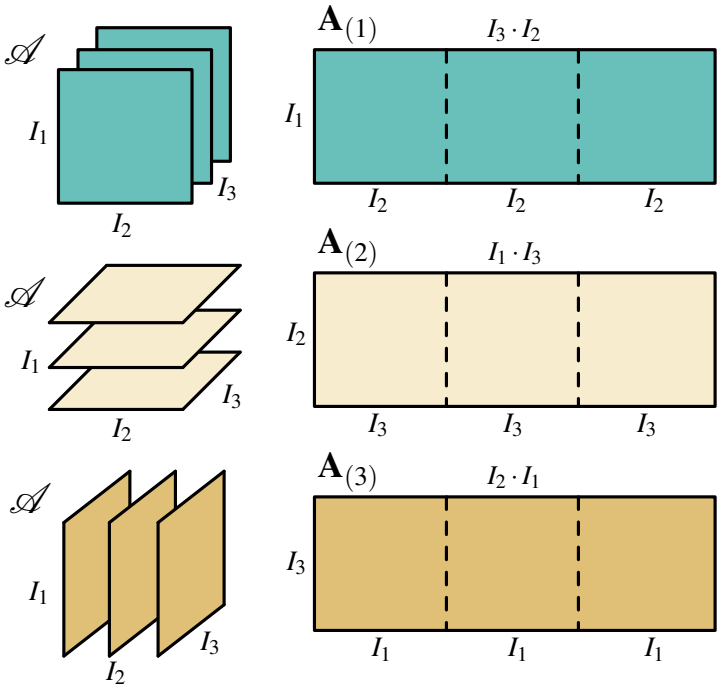
Rank of a Tensor

In order to describe the definitions of the tensor rank, the definition for the matrix rank is recaptured. The *matrix rank* of a matrix $\mathbf{A}$ is defined over its column and row ranks, i.e., the column and row *matrix rank* of a matrix $\mathbf{A}$ is the maximal number of linearly independent columns and rows of $\mathbf{A}$, respectively. For matrices, the column rank and the row rank are always equal and, a matrix rank is therefore simply denoted as $rank(\mathbf{A})$. A *tensor rank* is defined similarly to the matrix rank. However, there are differences. In fact, the extension of the rank concept is not uniquely defined in higher-orders. The definitions for the tensor ranks are taken from [De Lathauwer et al., 2000a].

- The *n-rank* of a tensor $\mathscr{A}$, denoted by $R_n = rank_n(\mathscr{A})$, is the dimension of the vector space spanned by mode-n vectors, where the mode-n vectors of $\mathscr{A}$ are the column vectors of the unfolding $\mathbf{A}_{(n)}$, and $rank_n(\mathscr{A}) =$



(a) backward cyclic unfolding [De Lathauwer et al., 2000a]



(b) forward cyclic unfolding [Kiers, 2000]

Figure 3.11: *Backward vs. forward cyclic unfolding of a tensor3.*

$rank(\mathbf{A}_{(n)})$. Unlike matrices, the n-ranks of a tensor are not necessarily the same.

- A higher-order tensor has a *multilinear rank* $(R_1, R_2, \ldots, R_N)$ [Hitchcock, 1927a; Hitchcock, 1927b] if its mode-1 rank (row vectors), mode-2 rank (column vectors) until its mode-N rank are equal to $R_1, R_2, \ldots, R_N$, e.g., a multilinear rank-(R_1, R_2, R_3) for a 3^{rd}-order tensor.

- A *rank-one tensor* is an N-way tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ under the condition that it can be expressed as the outer product of N vectors, as in Eq. (3.10) (see also [Kruskal, 1989; Comon and Mourrain, 1996]). A rank-one tensor is also known under the term *Kruskal tensor*.

$$\mathscr{A} = \boldsymbol{b}^{(1)} \circ \boldsymbol{b}^{(2)} \circ \cdots \circ \boldsymbol{b}^{(N)} \tag{3.10}$$

- The *tensor rank* $R = rank(\mathscr{A})$ is the minimal number of rank-one tensors that yield $\mathscr{A}$ in a linear combination (see [Kruskal, 1989; Comon and Mourrain, 1996; De Lathauwer et al., 2000a; Kolda and Bader, 2009]). Except for the special case of matrices, the tensor rank is not necessarily equal to any of its n-ranks. It always holds that $R_n \leq R$.

3.3.2 Higher-Order SVD (HOSVD)

The *HOSVD* or *multilinear SVD* [De Lathauwer et al., 2000a], which is a higher-order generalization of the SVD, is a basic algorithm that is used to compute the different tensor decomposition models. The idea of the HOSVD is to compute a matrix SVD along every mode of the input tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. To achieve this, the tensor $\mathscr{A}$ is unfolded along every mode n to its matrix representation $\mathbf{A}_{(n)}$, as shown in Fig. 3.11. Then a matrix SVD is computed on the unfolded matrix $\mathbf{A}_{(n)}$. The R_n leading left singular vectors are chosen as the basis $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for the mode n. As shown in Alg. 1, this procedure is repeated for every mode n.

Algorithm 1 HOSVD along every mode n.

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, $(R_1, R_2, \ldots, R_N)$
2: **Output:** $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$
3: **for** every mode n of N **do**
4: unfold $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into its matrix representation $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1} \cdot I_{n+1} \cdots I_N)}$
5: compute the matrix SVD $\mathbf{A}_{(n)} = \mathbf{U}^{(n)} \mathbf{\Sigma} \mathbf{V}^{(n)T}$
6: set the R_n leading left singular vectors to the mode-n basis $\mathbf{U}^{(n)}$
7: **end for**

3.4 Tensor Decompositions

In general, in tensor decompositions an input tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is decomposed into a set of factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ and coefficients that describe the relationship/interactivity between $\mathscr{A}$ and the set of $\mathbf{U}^{(n)}$.

Historically, as seen earlier, tensor decompositions are a higher-order extension of the matrix SVD/PCA. The nice properties of the matrix SVD, i.e., rank-R decomposition and orthonormal row-space vectors and column-space vectors, do not extend uniquely to higher orders. The rank-R decomposition can be achieved with the so-called CP model, while the orthonormal row and column vectors are preserved in the so-called Tucker model. An extensive review of the two models and further hybrid models can be found in [Kolda and Bader, 2009]. Here, we outline the two most common models, the Tucker model and the CP model. Hybrid models are mentioned only briefly.

3.4.1 Tucker Model

The Tucker model is a widely used approach for tensor decompositions. As given in Eq (3.11), any higher-order tensor is approximated by a product of a core tensor $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and its factor matrices $\mathbf{U}^{(\mathbf{n})} \in \mathbb{R}^{I_n \times R_n}$, where the products $\times_n$ denote the n-mode product (see App. C) between the tensor and the matrices. This decomposition can later be reconstructed to its approximation $\widetilde{\mathscr{A}}$. The missing information of the input tensor $\mathscr{A}$ that cannot be captured by $\widetilde{\mathscr{A}}$ is denoted with the error ε. The Tucker decomposition is visualized for a 3^{rd}-order tensor in Fig. 3.12. Alternatively, a Tucker decomposition can be expressed as a sum of rank-one tensors (Eq. (3.12) and Fig. 3.13).

$$\mathscr{A} = \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} + e \tag{3.11}$$

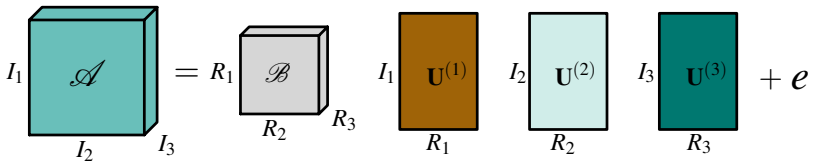


Figure 3.12: *Tucker tensor3:* $\mathscr{A} = \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)} + e$.

$$\mathscr{A} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_N=1}^{R_N} b_{r_1 r_2 \ldots r_N} \cdot \boldsymbol{u}_{r_1}^{(1)} \circ \boldsymbol{u}_{r_2}^{(2)} \circ \cdots \circ \boldsymbol{u}_{r_N}^{(N)} + e \tag{3.12}$$

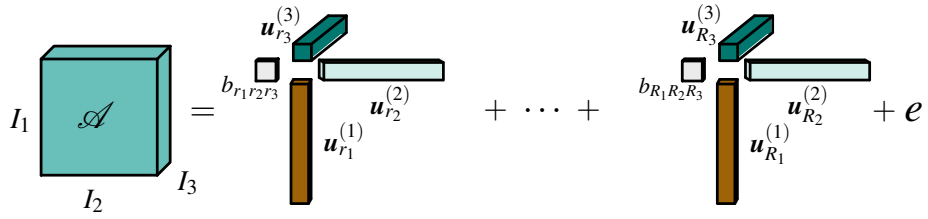


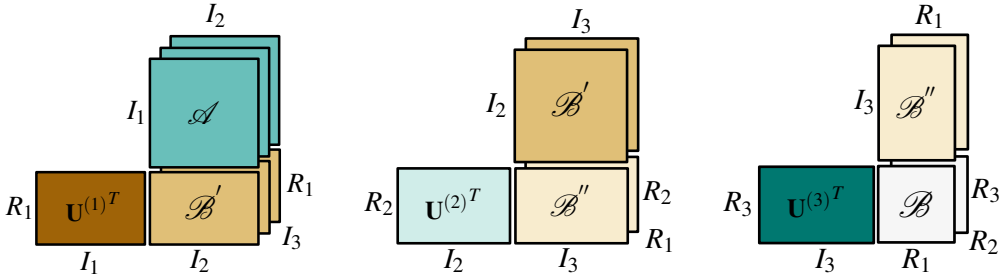
Figure 3.13: *Tucker tensor3 as a sum of rank-one tensors:* $\mathscr{A} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} b_{r_1 r_2 r_3} \cdot \mathbf{u}_{r_1}^{(1)} \circ \mathbf{u}_{r_2}^{(2)} \circ \mathbf{u}_{r_3}^{(3)} + e$.

The column vectors of the factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ are usually orthonormal and can be thought of as principal components R_n in each mode n [Kolda and Bader, 2009]. The core tensor $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ represents a projection of the original data $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ onto its factor matrices and is always of the same order as the input data. The core tensor is computed in general, as shown in Eq. (3.13), and for orthogonal factor matrices as in Eq. (3.14) (see Fig. 3.14). The element-wise core tensor computation is denoted in Eq. (3.15). In other words, the core tensor coefficients $b_{r_1 r_2 \ldots r_N}$ show the relationship or interactivity between the Tucker model and the original data.

$$\mathscr{B} = \mathscr{A} \times_1 \mathbf{U}^{(1)^{(-1)}} \times_2 \mathbf{U}^{(2)^{(-1)}} \times_3 \cdots \times_N \mathbf{U}^{(N)^{(-1)}} \tag{3.13}$$

$$\mathscr{B} = \mathscr{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \times_3 \cdots \times_N \mathbf{U}^{(N)^T} \tag{3.14}$$

$$\mathscr{B} = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} a_{i_1 i_2 \ldots i_N} \cdot \boldsymbol{u}_{i_1}^{(1)^T} \circ \boldsymbol{u}_{i_2}^{(2)^T} \circ \cdots \circ \boldsymbol{u}_{i_N}^{(N)^T} \tag{3.15}$$



(a) core TTM along mode 1 (b) core TTM along mode 2 (c) core TTM along mode 3

Figure 3.14: *Forward cyclic tensor times matrix (TTM) computation after [Kiers, 2000] in order to produce the core tensor* $\mathscr{B}$*: n-mode products along the three modes.*

The Tucker decomposition is not unique, which means that we can modify the core tensor $\mathscr{B}$ without affecting the model fit as long as we apply the same changes to the factor matrices (so-called core tensor rotations). This provides the option to rearrange the core tensor such that, for example, more values are zero. For details see [Kolda and Bader, 2009].

3.4.2 CP Model

The parallel factor analysis (PARAFAC) or the canonical decomposition (CANDECOMP), called CP in short, factorizes a tensor into a sum of R rank-one tensors. Hence, a tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ can be rank decomposed as a sum of R rank-one tensors as in Eq. (3.16). An example of a 3^{rd}-order CP decomposition is illustrated in Fig. 3.15. Note: The column vectors of the matrices in Eq. (3.16) are normalized, which yields a weighting factor λ_r for each term. The information not captured by the CP model is represented with the error ε.

$$\mathscr{A} = \sum_{r=1}^{R} \lambda_r \cdot \boldsymbol{u}_r^{(1)} \circ \boldsymbol{u}_r^{(2)} \circ \cdots \circ \boldsymbol{u}_r^{(N)} + e \tag{3.16}$$

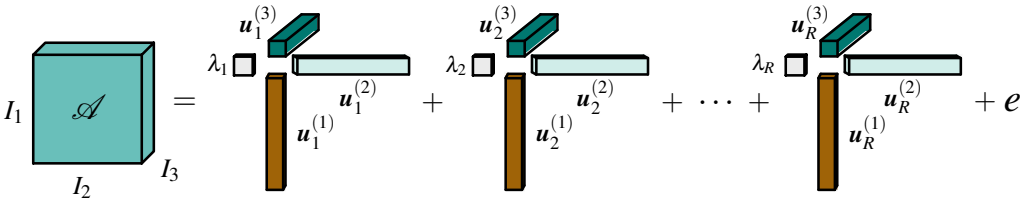


Figure 3.15: *CP tensor3, sum of rank-one tensors:* $\mathscr{A} = \sum_{r=1}^{R} \lambda_r \cdot \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \mathbf{u}_r^{(3)} + e$.

The CP model is in fact a special case of the Tucker model. The vector containing the λ-values can be arranged as the super-diagonal of a Tucker core tensor with R diagonal values, while the rest of the core tensor is zero (see Fig. 3.16). In contrast to the Tucker model, the CP model is unique under certain constraints (see [Kolda and Bader, 2009]). In this context, uniqueness means that the current CP model is the only possible combination of rank-one tensors that sums to $\widetilde{\mathscr{A}}$. However, permutation freedom and scaling is still possible.

3.4.3 Other Models

There are a number of other models, mostly some hybrid forms of the CP model and the Tucker model. One such model is the so-called *block-diagonal tensor decomposition* by [De Lathauwer, 2008a; De Lathauwer, 2008b; De Lathauwer and Nion, 2008], which produces a super-diagonal of P core tensor with zeros

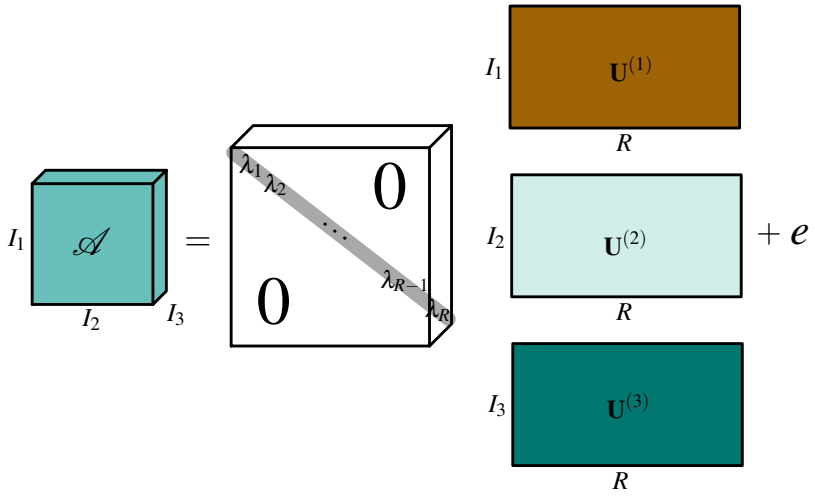


Figure 3.16: *CP tensor3 visualized as special case of a Tucker tensor3.*

except for the blocks forming the diagonal, as illustrated in Fig. 3.17. Other hybrid models can be found in the extensive review by [Kolda and Bader, 2009].

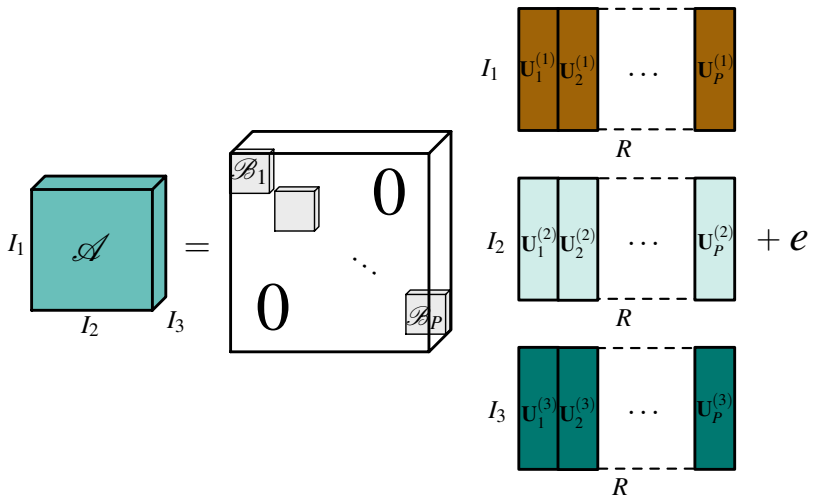


Figure 3.17: *Block-diagonal tensor3.*

Often, we are interested in compact models, which enable a compression of the input dataset. For example, after computing a Tucker decomposition by a HOSVD the core tensor $\mathscr{B}$ has the same size as the original input dataset $\mathscr{A}$ and all the factor matrices are quadratic. However, we are more interested in light-weight, approximative Tucker decompositions, where $\mathscr{B}$ is an element of $\mathbb{R}^{R_1 \times R_2 \times R_3}$ with $R_1 < I_1$, $R_2 < I_2$ and $R_3 < I_3$. Using so-called *rank-reduced tensor decompositions* or *truncated tensor decompositions* one can directly obtain more compact

decompositions. Furthermore, the truncated decompositions are usually better in terms of the difference between approximated and original data [Kolda and Bader, 2009]. In the next section, the tensor rank approximations corresponding to the Tucker model and the CP model are defined.

3.5 Tensor Rank Truncation

As seen in Sec. 3.3.1, the extension of the matrix rank concept to higher orders is not unique. There are two main directions followed, which are based on either a rank-one, i.e., a rank-R tensor decomposition or a rank-$(R_1, R_2, \ldots, R_N)$ tensor decomposition. Their rank-reduced approximations are defined accordingly:

i). A rank-one approximation is defined as $\widetilde{\mathscr{A}} = \lambda \cdot \boldsymbol{u}^{(1)} \circ \boldsymbol{u}^{(2)} \cdots \circ \boldsymbol{u}^{(N)}$ from the rank-one tensor (vector) product ($\circ$) of its basis vectors $\boldsymbol{u}^{(n)} \in \mathbb{R}^{I_n}$ and the scalar λ. Hence a tensor $\mathscr{A}$ can be approximated by a linear combination of rank-one approximations as in Eq. (3.17). This approximation, previously defined as a CP model, and is called a *rank-R approximation*.

$$\widetilde{\mathscr{A}} \approx \sum_{r=1}^{R} \lambda_r \cdot \boldsymbol{u}_r^{(1)} \circ \boldsymbol{u}_r^{(2)} \circ \cdots \circ \boldsymbol{u}_r^{(N)} \tag{3.17}$$

ii). Alternatively, a *rank-*$(R_1, R_2, \ldots R_N)$ *approximation* of $\mathscr{A}$ is formulated as a decomposition into a lower-rank tensor $\widetilde{\mathscr{A}} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ with $rank_n(\widetilde{\mathscr{A}}) = R_n \leq rank_n(\mathscr{A})$. The approximated tensor is the n-mode product $\times_n$ of factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ and a core tensor $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \cdots \times R_N}$ in a given reduced rank space (Eq. (3.18)). This rank-$(R_1, R_2, \ldots R_N)$ approximation was previously introduced as the *Tucker* model.

$$\widetilde{\mathscr{A}} \approx \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} \tag{3.18}$$

In general a rank-reduced approximation is sought such that the least-squares cost function in Eq. (3.19) is minimized.

$$\widetilde{\mathscr{A}} = \arg\min(\widetilde{\mathscr{A}}) \left\| \mathscr{A} - \widetilde{\mathscr{A}} \right\|^2 \tag{3.19}$$

The notation of the different rank-approximations becomes interesting for compression approaches. Given that $(R_1, R_2, \ldots R_N)$ or R are sufficiently smaller than the initial lengths $(I_1, I_2, \ldots, I_N)$, the coefficients $\Lambda \in \mathbb{R}^R$ or $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and the factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ lead to a compact approximation of $\widetilde{\mathscr{A}}$ of

the original tensor $\mathscr{A}$. In particular, the multilinear rank-$(R_1, R_2, \ldots R_N)$ is typically explicitly chosen to be smaller than the initial ranks in order to achieve a compression on the input data. In contrast, the CP model often needs larger factor matrices, where often $R_n \gg I_n$ is necessary to represent the dataset (see Fig. 3.16).

Choosing principal components In tensor approximation, we would like to make use of selecting major components from the decomposition as is similarly known from the matrix PCA. That is, by eliminating the higher-ranked principal components and their basis vectors, we preserve the most important direction-s/structures in the dataset. In other words, this means that we have reconstructed the major components of the original datasets but that some details are missing. These details can be added by progressively reconstructing more and more principal components to the approximated form of the original dataset. In practice, many of the insignificant principal components or their basis vectors are very low or close to zero, i.e., they are negligible. Typically, the first couple of principal components already define most of the total variability within a dataset. For data approximation techniques, we therefore often use only a certain number of principal components and their basis vectors to represent a dataset, i.e., we work with a reduced set of singular values σs and truncated factor matrices (see App. B). Correspondingly, a rank-reduced or truncated tensor decomposition is desired.

Truncated tensor decompositions The tensor rank parameter R_n is responsible for the number of TA coefficients and bases that are used for the reconstruction and hence is responsible for the approximation level. In higher orders, the CP decomposition produced from an alternating least squares (ALS) algorithm (see App. D), can not be truncated per se. The ex post truncation of the Tucker decomposition, however, is possible due to the *all-orthogonality* property of the core tensor. For a 3^{rd}-order tensor, all-orthogonality means that the different horizontal matrices of the core $\mathscr{B}$ (the first index i_1 is kept fixed, while the two other indices, i_2 and i_3, are free) are mutually orthogonal with respect to the scalar product of matrices (i.e., the sum of the products of the corresponding entries vanishes). The same holds for fixed indices i_2 and i_3 (see [De Lathauwer et al., 2000a]). Therefore, given an initial rank-(R_1, R_2, R_3) Tucker model, we can progressively choose lower ranks $K_n < R_n$ for reduced quality reconstruction. As indicated in Fig. 3.18 on the example of the Tucker model, the rank indicates how many factor matrix columns and corresponding core tensor entries are used for the reconstruction. From that, we conclude that there are basically two ways to go: (1) either start with the desired rank truncation as initially described or (2) subsequently or progressively truncate the given decomposition.

As in the matrix PCA case, a small R_n corresponds to a low-rank Tucker tensor

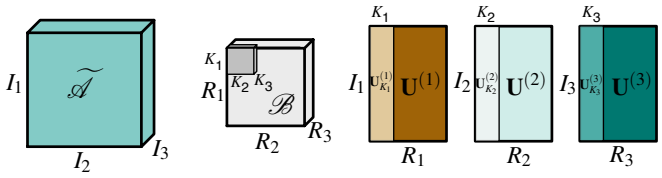


Figure 3.18: *Illustration of a truncated Tucker tensor reconstruction: A reduced range of factor matrix columns with corresponding fewer core tensor entries reconstructs a lower quality approximation but at full resolution.*

approximation (many details removed) and a large R_n corresponds to an approximation matching the original more closely. The ordering of the coefficients in the core tensor is not strictly decreasing as in the matrix SVD case the singular values are; however, in practice, it can be shown that progressive tensor rank truncation in the Tucker model works well for adaptive visualization of the data at different feature scales.

The algorithms to compute such truncated tensor decompositions are summarized in App. D. After having introduced the concepts for (truncated) tensor decompositions, we look at the possible reconstruction approaches. In fact, it is critical to choose the appropriate reconstruction approach in order to achieve a real-time reconstruction for interactive visualization.

3.6 Tensor Reconstruction

The *tensor reconstruction* of a reduced-rank Tucker decomposition can be achieved in different ways. One alternative is a progressive reconstruction: Each entry in the core tensor $\mathscr{B}$ is considered as weight for the outer product between the corresponding column vectors in the factor matrices. This looks like Eq. (3.20) for the Tucker reconstruction and Eq. (3.17) for the CP reconstruction.

$$\widetilde{\mathscr{A}} \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_N=1}^{R_N} b_{r_1 r_2 \dots r_N} \cdot \boldsymbol{u}_{r_1}^{(1)} \circ \boldsymbol{u}_{r_2}^{(2)} \circ \dots \circ \boldsymbol{u}_{r_N}^{(N)} \tag{3.20}$$

This reconstruction strategy corresponds to reconstructing rank-one tensors and cumulatively summing them up. The weighted "subtensors" then form the approximation $\widetilde{\mathscr{A}}$ of the original data $\mathscr{A}$. In particular for the Tucker model, this is an expensive reconstruction strategy since it involves multiple for-loops to run over all the summations, which typically slows down the computing time.

3.6.1 Element-wise Reconstruction

Another approach, is to reconstruct each element of the approximated dataset individually, which we call *element-wise reconstruction* approach. Each element $\widetilde{a}_{i_1 i_2 i_3}$ is reconstructed, as shown in Eq. (3.21) for the Tucker reconstruction, and as shown in Eq. (3.22) for the CP reconstruction. For the Tucker model that is: All core coefficients multiplied with the corresponding coefficients in the factor matrices are summed up (weighted sum). Similarly, this applies for the CP model expect that we have only diagonal core coefficients or λ's as they are usually called.

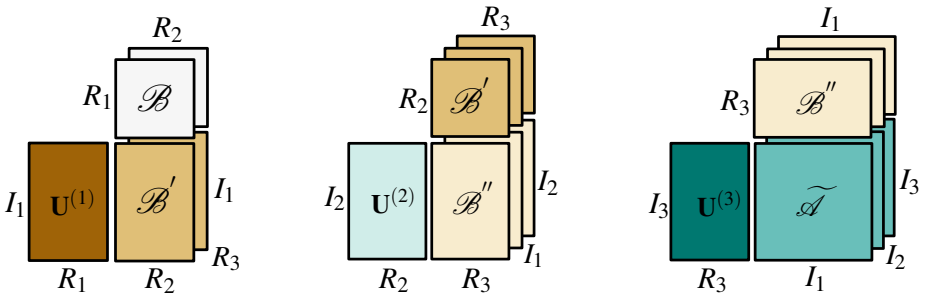
$$\widetilde{a}_{i_1 i_2 \ldots i_N} \approx \sum_{r_1 r_2 \ldots r_N} b_{r_1 r_2 \ldots r_N} \cdot u^{(1)}_{i_1 r_1} \cdot u^{(2)}_{i_2 r_2} \cdot \cdots \cdot u^{(N)}_{i_N r_N} \tag{3.21}$$

$$\widetilde{a}_{i_1 i_2 \ldots i_N} \approx \sum_{r=1}^{R} \lambda_r \cdot u^{(1)}_{i_1 r} \cdot u^{(2)}_{i_2 r} \cdot \cdots \cdot u^{(N)}_{i_N r} \tag{3.22}$$

The element-wise reconstruction can be beneficial for applications where only a sparse amount of reconstructed elements are needed.

3.6.2 Optimized Tucker Reconstruction

A third reconstruction approach – applying only to the Tucker reconstruction – is to build the n-mode products along every mode, which leads to a tensor times matrix (TTM) multiplication for each mode, e.g., TTM1 along mode 1, (see Eq. (C.3) and Fig. C.2). This is analogous to the Tucker model given by Eq. (3.18). In Fig. C.2 we visualize the TTM reconstruction applied to a 3^{rd}-order tensor using n-mode products.



(a) TTM 1 (b) TTM 2 (c) TTM 3

Figure 3.19: *Forward cyclic TTM multiplications after [Kiers, 2000] along the three modes (n-mode products).*

Given the fixed cost of generating a $I_1 \times I_2 \times I_3$ grid, the computational overhead factor varies from cubic rank complexity $R_1 \cdot R_2 \cdot R_3$ in the case of the progressive reconstruction (Eq. (3.20)) to a linear rank complexity R_1 for the TTM or the n-mode product reconstruction (Eq. (3.21)). For example, for $R = 16$, the improvement to $R^3 = 4'096$ is 256-fold.

3.6.3 CP Reconstruction by the Khatri-Rao Product

For completeness, an alternative CP reconstruction strategy is mentioned as well. That is, the CP reconstruction can be computed with the Khatri-Rao product $\odot$ (see App. C), as in the example of a 3^{rd}-order tensor in Eq. (3.23).

$$\widetilde{\mathbf{A}}_n \approx \mathbf{U}^{(1)} \left(\mathbf{U}^{(2)} \odot \mathbf{U}^{(3)}\right)^T \tag{3.23}$$

However, it is to be noted that this reconstruction strategy produces large matrices (see Sec. 3.3.1) due to the Khatri-Rao product, which extends the matrix rows and the matrix columns to a multiple between the rows and the columns of two matrices, respectively. Obviously, that results in more expensive matrix-matrix multiplications.

C H A P T E R 4

RESULTS: MULTIRESOLUTION MODELING WITH TA

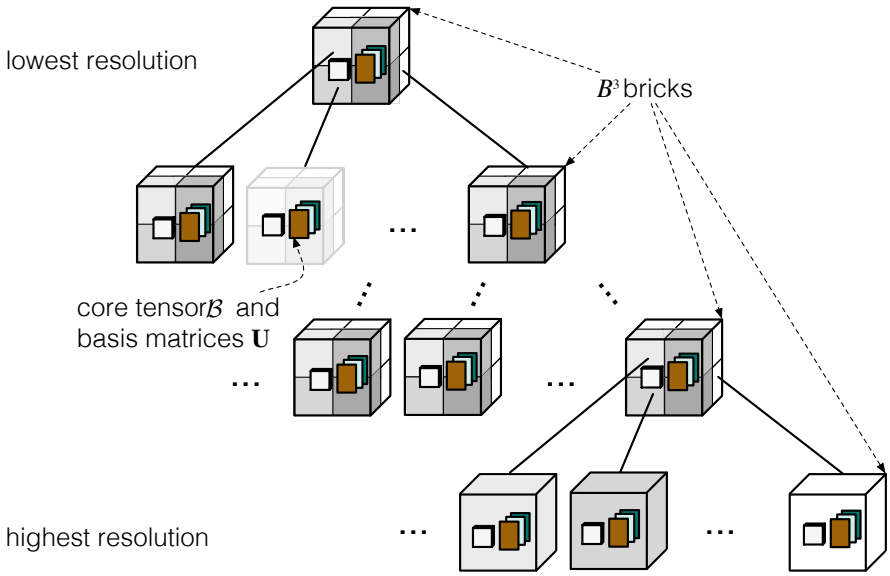
4.1 Two Multiresolution Models

As discussed in Sec. 3.2 a multiresolution model is an essential part for the volume rendering of large datasets. The chosen method is based on the offline decomposition of the original volumetric dataset into small cubical bricks (subvolumes), which are organized into an octree structure maintained out-of-core. The octree contains data bricks at different resolutions, where each resolution of the volume is represented as a collection of bricks in the subsequent octree hierarchy level. In this thesis, two multiresolution models based on compact data representation by tensor approximation were developed. Both models are organized into a hierarchical octree, which contains per octree node a brick of the original dataset. In the first model, each octree node represents an own local tensor decomposition (Fig. 4.1(a)). In the second model, each node contains only a core tensor, which represents the coefficients for the current brick relative to global tensor bases, which are valid for the full volume (Fig. 4.1(b)). The second model, in contrast to the first model, not only stores local brick information in the bases, but also captures the global energy distribution of the dataset, which will then be projected on and stored in the brick-wise core tensors.

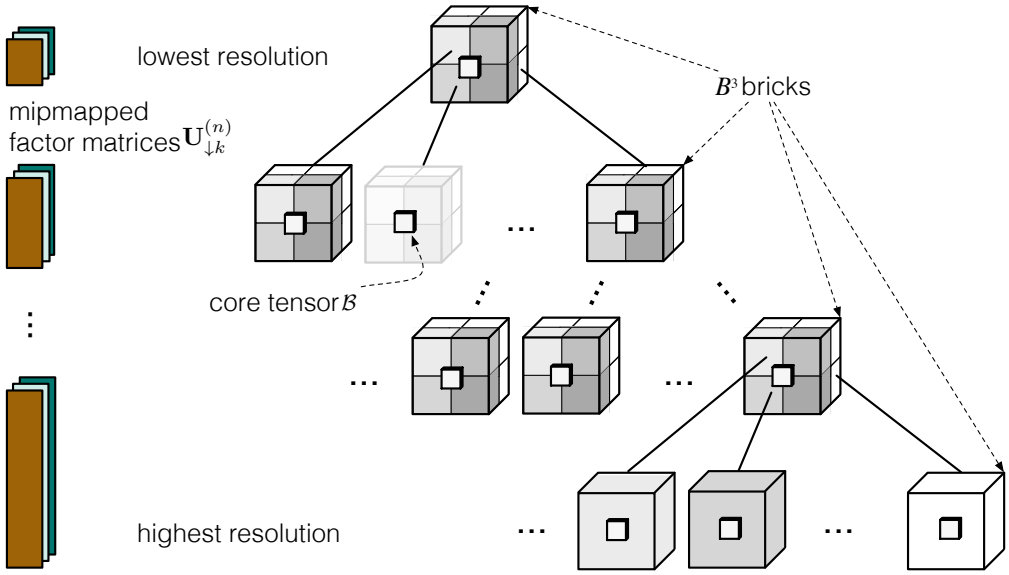
For both multiresolution models, we chose the Tucker model (Sec. 3.4.1) as an underlying tensor approximation model. In a study on the applicability of TA to interactive volume visualization [Ballester-Ripoll et al., 2013], the Tucker model has been shown to capture the volumetric datasets in the most compact way. Moreover, the same study showed that the truncation of coefficients in the Tucker model works best. Regarding the random-access of individual voxels, the Tucker model makes it possible (see Eq. (3.21)) to reconstruct individual elements. This makes the direct rendering from coefficients of the decomposition possible. However, the voxel-wise reconstruction is not necessarily the fastest approach. There are tensor reconstruction approaches, which reconstruct a full brick with reduced computational complexity (Sec. 3.6.2).

With respect to the rendering, each subvolume or brick in the octree hierarchy has a fixed width B with an overlap of two voxels at each brick boundary to efficiently support run-time operations that require access to neighboring voxels (trilinear interpolation and gradient computation). In application 1 a brick size of $B = 32$ was chosen, while in application 2 the brick size was increased to $B = 64$.

The applicability of the two TA multiresolution DVR models is discussed in Sec. 4.2 and Sec. 4.4 and was published in [Suter et al., 2011] and [Suter et al., 2013], respectively. TA properties used for the second application with the global TA bases are elaborated in Sec. 4.3.



(a) multiresolution model with per brick TA bases



(b) multiresolution model with global TA bases

Figure 4.1: *Two multiresolution octree decomposition hierarchies with B^3 sized data blocks: (a) model with per brick tensor decompositions (adapted from [Suter et al., 2011]), (b) model with global TA bases and per brick core tensors [Suter et al., 2013].*

4.2 Application 1: Brick-wise TA Bases Model

In this first application as described in [Suter et al., 2011], each octree node is represented by its own local tensor decomposition applied to the respective data brick. The brick width is set to $B = (28+2+2) = 32$, i.e., one brick is 32^3, which has proved small enough to guarantee level-of-detail (LOD) adaptivity, while coarse enough to permit an effective brick encoding by the analysis of the local structure.

Each octree brick $\mathscr{A}_{brick} \in \mathbb{R}^3$ is approximated using rank-reduced Tucker decompositions. A Tucker decomposition (see Sec. 3.4) is defined as $\widetilde{\mathscr{A}} = \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}$, where $\mathscr{B}$ is the so-called core tensor and $\mathbf{U}^{(n)}$ are the factor matrices. A rank-reduced TA along every mode of the dataset is written with the notation: rank-(R_1, R_2, R_3) TA. As illustrated in Fig. 4.1(a), we compute for each brick of size B^3 a rank-(R,R,R) TA, with $R \in [1..B-1]$. An initial rank reduction, where $R = B/2$, i.e., $R = 16$ for $B = 32$, is used following the rank reduction scheme used in other tensor approximation works [Wu et al., 2008; Suter et al., 2010a]. After the decomposition, the tensor coefficients are stored in a quantized version. As suggested in [Suter et al., 2011], a 16-bit factor matrix encoding and an 8-bit core tensor encoding was applied.

The whole preprocessing is performed in a low-memory setup using a bottom-up process on a brick-by-brick basis, which is repeated until the octree root is reached. Leaves are constructed by sampling the original dataset, while non-leaf bricks are constructed from their previously constructed eight children, which are reconstructed, and spatially averaged.

For instance, the veiled chameleon dataset is visualized with the presented out-of-core multiresolution volume renderer based on tensor reconstructed bricks, as shown in Fig. 4.2.

Figure 4.2: *Chameleon dataset rendered with the per brick multiresolution TA model, from [Suter et al., 2011].*

4.3 Useful TA Properties for Multiresolution DVR

Historically, tensor approximation is a higher-order extension of the SVD or PCA. The nice properties of the matrix SVD, i.e., rank-R decomposition and orthonormal row-space and column-space vectors, do not fully extend to higher orders. The rank-R decomposition can be achieved with the so-called CP model, while the orthonormal row and column vectors are preserved in the so-called Tucker model. In the following, we refer to the Tucker model and decomposition.

The Tucker model (Sec. 3.4.1) consists of one factor matrix per mode (data direction) $\mathbf{U}^{(\mathbf{n})} \in \mathbb{R}^{I_n \times R_n}$ and one core tensor $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \ldots R_N}$. The core tensor $\mathscr{B}$ is in effect a projection of the original data $\mathscr{A}$ onto the basis of the factor matrices $\mathbf{U}^{(\mathbf{n})}$. In case of a volume, the Tucker model has three modes, as illustrated in Fig. 3.12, and defines an approximation $\widetilde{\mathscr{A}} = \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}$ of the original volume $\mathscr{A}$ (using n-mode products $\times_n$).

The row and column axes of the factor matrices represent two different spaces: (1) the rows correspond to the spatial dimension in the corresponding mode, and (2) the columns to the approximation quality. In the following we show how these properties can be exploited for multiresolution modeling (spatial selection and subsampling of rows) along the vertical axis (see Fig. 4.3). The concepts of spatial selectivity and spatial subsampling within TA bases as described in this section are published in [Suter et al., 2013]. The next two subsections not only elaborate the spatial properties of the TA matrices, but also illustrate how these properties can be exploited in multiresolution DVR.

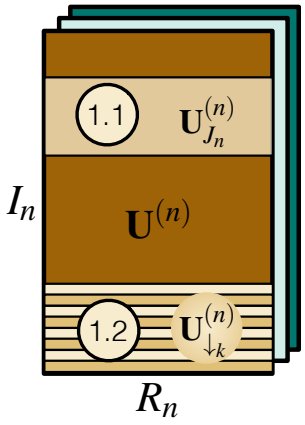


Figure 4.3: *Factor matrix properties along the vertical axis: (a) spatial selectivity and (b) spatial subsampling.*

4.3.1 Spatial Selectivity

For view-frustum culling and adaptive brick selection in interactive multiresolution volume visualization, efficient access to spatially restricted subvolumes is required. Since a TA factor matrix's rows directly correspond to its spatial dimension, we can exploit this fact for the reconstruction of a subvolume directly from the global factor matrices. We first describe the spatial selection for a given fixed resolution and explain the multiresolution access in the following section.

The Tucker model defines an approximation of a volume $\mathscr{A}$ by the decomposition $\widetilde{\mathscr{A}} = \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}$, and each element of $\widetilde{\mathscr{A}}$ is defined as

$$\widetilde{a}_{i_1 i_2 i_3} = \sum_{r_1} \sum_{r_2} \sum_{r_3} b_{r_1 r_2 r_3} \cdot u^{(1)}_{i_1 r_1} \cdot u^{(2)}_{i_2 r_2} \cdot u^{(3)}_{i_3 r_3}, \tag{4.1}$$

with factor matrix and core tensor entries $u^{(n)}_{i_n r}$ and $b_{r_1 r_2 r_3}$ (see also [Kolda and Bader, 2009]).

Due to the correspondence of the rows of $\mathbf{U}^{(n)}$ to the spatial dimension n (see Fig. 4.3), we can define row-index subranges $J_n \subseteq [0 \ldots I_n]$ that reconstruct a well defined spatial subvolume $J_1 \times J_2 \times J_3$ for the reduced index ranges $i_n \in J_n$ in Eq. 4.1. As illustrated in Fig. 4.4, we can thus select and reconstruct a subvolume of the dataset, e.g., corresponding to an octree brick, by choosing a subset of the row vectors of all factor matrices. Using these row-block submatrices $\mathbf{U}^{(n)}_{J_n}$ we can formulate the subvolume reconstruction as

$$\widetilde{\mathscr{A}}_{J_1 \times J_2 \times J_3} = \mathscr{B} \times_1 \mathbf{U}^{(1)}_{J_1} \times_2 \mathbf{U}^{(2)}_{J_2} \times_3 \mathbf{U}^{(3)}_{J_3}. \tag{4.2}$$

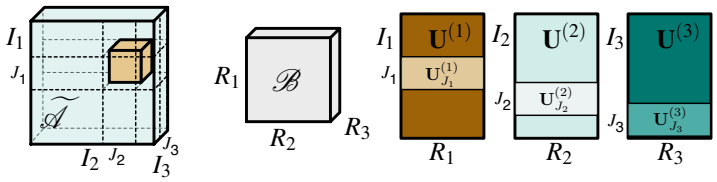


Figure 4.4: *Illustration of spatial selectivity within TA bases: A range of selected submatrix rows reconstructs a well defined subvolume (in brown) of the original whole dataset. From [Suter et al., 2013].*

4.3.2 Spatial Subsampling

As outlined in the introduction, in interactive multiresolution volume visualization, we need lower resolution subsampled and averaged representations of subvolume bricks for view-dependent adaptive LOD rendering. Due to the direct

spatial correspondence of factor-matrix rows to the spatial dimensions as outlined above, we can apply the lower-resolution subsampling on factor matrices before brick reconstruction from the TA representation.

Since the I_n rows of a factor matrix $\mathbf{U}^{(n)}$ correspond to the resolution of the volume $\widetilde{\mathscr{A}}$ in that mode, we can construct a lower-resolution reconstruction in the n-th dimension by first merging and averaging (pairs of) rows to get a downsampled matrix $\mathbf{U}^{(n)}_{\downarrow 1}$ (with $I_n/2$ rows). This is possible because the columns of a factor matrix capture the data variation along that dimension. Therefore, downsampling and averaging pairs of rows correspond to halving the reconstructed volume resolution. This downsampling of factor matrices is indicated in Fig. 4.5 and corresponds to what is known as *mipmapping*.

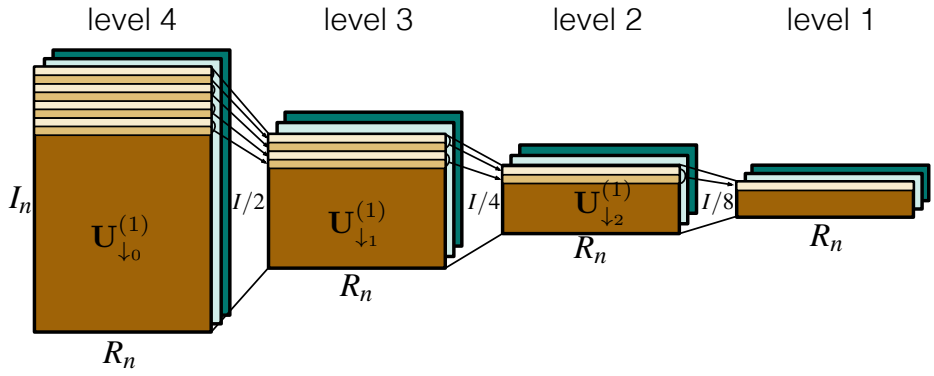


Figure 4.5: *Mipmapping of the global factor matrices: Subsampling by averaging (example input data is of size* 512^3 *and the bricks are of size* 64^3*), from [Suter et al., 2013].*

4.4 Application 2: Model with Global TA Bases

The second TA hierarchy, as illustrated in Fig. 4.1(b), follows again the principle of an octree subdivision of the input volume dataset and is published in [Suter et al., 2013]. A key part of this approach unlike any other TA proposed in visual computing before, is that a global set of mipmapped factor matrices $\mathbf{U}^{(n)}_{\downarrow k}$ that capture in every brick some global dataset information is maintained. Thus all nodes on one level l of the octree hierarchy are reconstructed using the same factor matrices $\mathbf{U}^{(n)}_{\downarrow k}$ corresponding to that octree level (with $k = l_{max} - l$). Unlike the first model, in this second model the core tensor(s) capture also some global information, since the factor matrices were not derived from the octree brick only, but rather on the full input volume. Still, the core tensor(s) hierarchy is necessary to be defined.

In practice, we cannot keep one large global core tensor $\mathscr{B}$ and selectively reconstruct individual octree nodes due to the fact that the core tensor entries, un-

like the factor matrix rows, do not exhibit any mapping to the spatial dimensions. Therefore, we define a small core tensor $\mathscr{B}_{\text{brick}}$ per octree node. Since octree nodes correspond to subvolume bricks, and practically in the described application a brick size $B = 64^3$ was used, the per-brick rank can be set accordingly to or less than half of the brick dimension, i.e. $R_n \leq 32$ (as motivated in [Wu et al., 2008; Suter et al., 2010a; Suter et al., 2011]).

Eventually, the core tensors are stored in quantized form (the 32-bit floating point coefficients are quantized with logarithmic step sizes to 8-bit values, as described in [Suter et al., 2011]). Our TA hierarchy is hence defined by a set of global factor matrices $\mathbf{U}^{(n)}_{\downarrow k}$ and an octree hierarchy that stores one small quantized core tensor $\mathscr{B}_{\text{brick}}$ of size 32^3 per node. Reconstruction is possible in a flexible way, according to a desired spatial resolution by choosing the octree level, and adapting the approximation scale by adjusting the rank reduction level $8 \leq R_n \leq 32$ (ranks less than 8 have been shown not to provide useful reconstructions).

For interactive visualization, the reconstruction from the multiresolution model is done in a similar way to [Suter et al., 2011]. The main difference of the presented TA hierarchy is that the global TA factor matrices are permanently loaded onto the GPU, and can be accessed from fast read-only graphics memory. For each brick, the corresponding core tensor, its scale factor, its rank, its hierarchy level and the brick's spatial position in the dataset are uploaded on demand onto the GPU. The core tensors are decompressed on demand once the corresponding bricks at the given LOD are requested. The decoding is performed using consecutive tensor times matrix multiplications, as shown in [Suter et al., 2011], but using the global factor matrices hierarchy.

The performed experiments performed show that it is feasible, first, to decompose large initial factor matrices of volume datasets, and second, to reconstruct the volumes at multiple resolutions by subsampling of the large initial factor matrices (see Figs. 4.6–4.8).

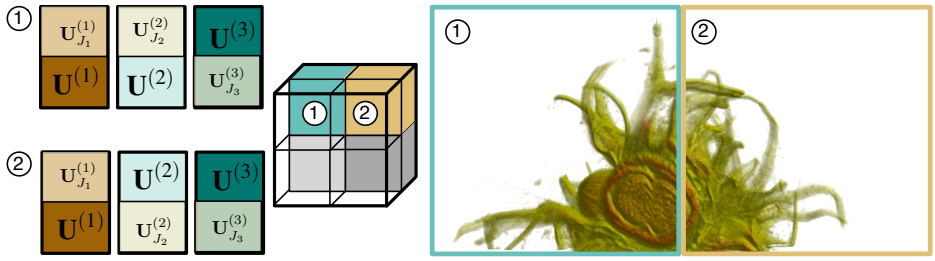
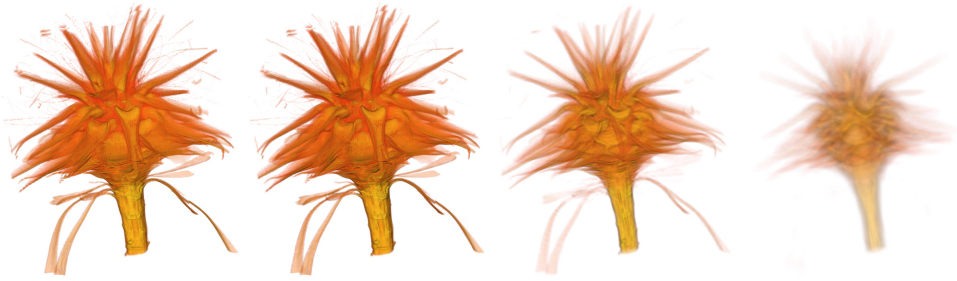


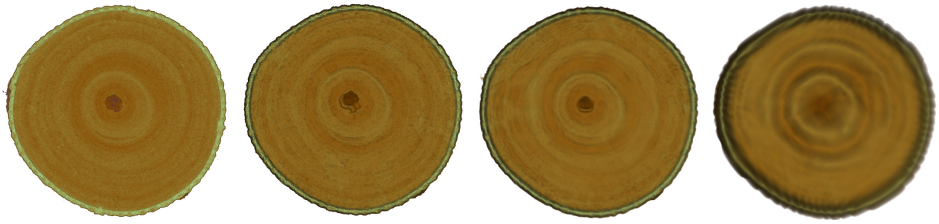
Figure 4.6: *Spatial selectivity of factor matrices. Two selected bricks are reconstructed by the corresponding selection of row index subranges. From [Suter et al., 2013].*

In order to have smoother brick transitions during volume rendering, we ap-

Figure 4.7: *Factor matrix subsampling (bottom row) compared to direct TA (middle row) derived from original subsampled input datasets (top row). From [Suter et al., 2013].*



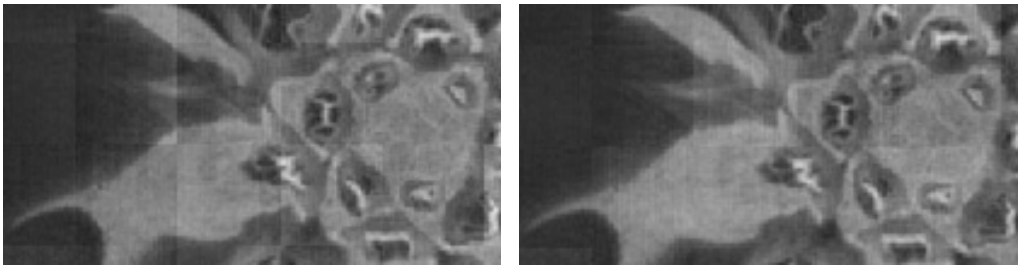
(a) original (b) level 5 (c) level 3 (d) root



(e) original (f) level 5 (g) level 3 (h) root

Figure 4.8: *Different spatial resolution levels reconstructed from the global TA bases: (a-d) flower dataset of size* 1024^3*, (e-h) wood branch of size* 2048^3*. The brick size is 64 for both datasets. From [Suter et al., 2013].*

plied two sorts of brick borders. Notably, we chose a 2-voxel border for the gradient interpolation and an additional 4-voxel border to incorporate neighboring brick information in the core tensor (see Fig. 4.9).



(a) no brick borders (b) with bricks borders

Figure 4.9: *A slice through the reconstructed flower dataset once without (a) and once with additional brick borders (b) from the global TA bases. From [Suter et al., 2013].*

4.5 Summary

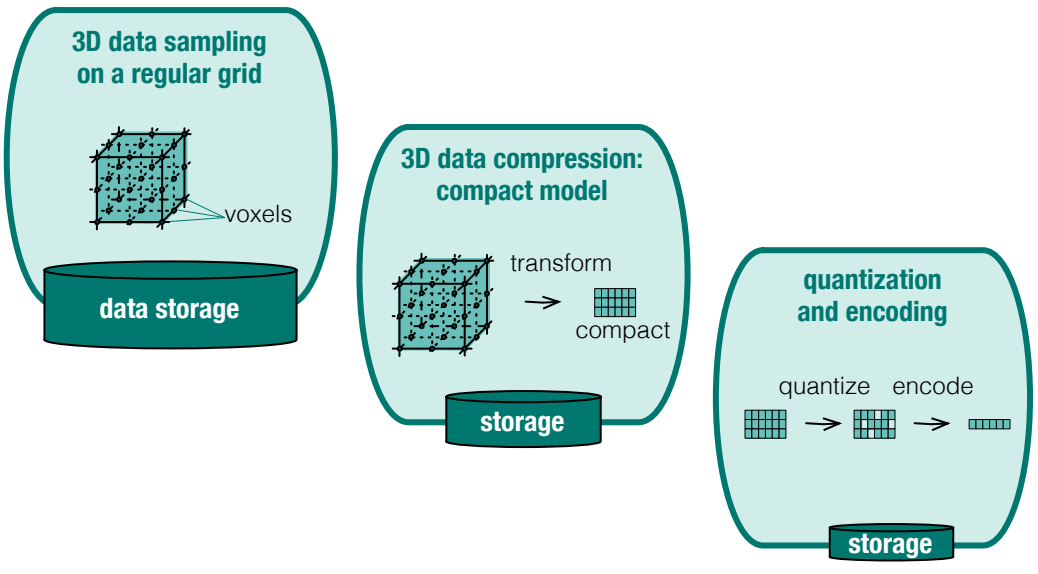
In this chapter, it was shown that TA is a suitable framework for multiresolution DVR. Two different Tucker tensor-based multiresolution models were developed and applied to real datasets. The main difference between the two models is the setup of the factor matrices. One model shows an octree hierarchy with bricked tensor decompositions of the local bricks corresponding to octree nodes, while the other model shows an octree hierarchy of only core tensors and the factor matrices were generated from the full input volume and therefore additionally store global information in every data brick. The second model, furthermore, exploits properties along the spatial dimension of the TA factor matrix bases. The spatial selectivity and the spatial subsampling within TA bases matches well the nature of multiresolution models. Spatial selectivity can be used for view-frustum culling and adaptive brick selection, and spatial subsampling can be used for the subsampled/averaged lower-resolution representation of the full dataset.

In contrast to other hierarchical tensor models, e.g., [Wu et al., 2008], we only applied direct tensor decompositions. In other words, this means that we explicitly chose not to use incremental hierarchical tensor models, which encode the data distributed over different resolution levels. We considered it a disadvantage to use an incremental hierarchical octree model since it implies an accumulated reconstruction process over several octree nodes.

Since one major goals was to achieve a storage reduced multiresolution model, the costs of TA multiresolution DVR models are evaluated next.

C H A P T E R 5

RESULTS: DATA REDUCTION AND COMPRESSION



5.1 Data Reduction and Compression with TA

As mentioned in the introduction and visualized on the cover picture of this chapter, data reduction and compression consist of several steps. First, the input data is transformed into a compact data representation, which consists of fewer coefficients than the original data. Second, the compact data representation is quantized (insignificant coefficients thresholded, floating point-integer conversion), and third, the data is encoded with a data stream encoding such as run-length encoding. In this thesis, we address the first two steps, the compact data representation transform and the TA-specific coefficient quantization. In order to judge the goodness of the selected data reduction approach, the ratio for the reduced data storage is evaluated. Other compression issues are related to a fast reconstruction or decoding and a fast access of individual coefficients. These latter aspects are covered in Chap. 7. In the following, we describe the general storage requirements of tensor approximation. Since in all applications of this thesis the Tucker model (Sec. 3.4.1) was used, the analysis for the requested storage size is also performed for the Tucker model.

Given an original volume, i.e., a 3^{rd}-order tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, the required storage for a Tucker tensor decomposition $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ and $\mathbf{U}^{(1..3)} \in \mathbb{R}^{I_n \times R_n}$ is $R_1 \cdot R_2 \cdot R_3 + I_1 \cdot R_1 + I_2 \cdot R_2 + I_3 \cdot R_3$. Since R_n is typically significantly smaller than I_n, often starting from $R_n \leq \frac{1}{2} \cdot I_n$ (see [Wu et al., 2008; Suter et al., 2010a; Suter et al., 2011]) the storage size is much reduced from the initial volume size of $I_1 \cdot I_2 \cdot I_3$. Replacing R_n by $\frac{1}{2} \cdot I_n$ and assuming that $I_1 \approx I_2 \approx I_3$ the storage cost for a Tucker decomposition of an initial volume I^3 is at least reduced to $\frac{3}{2} \cdot I^2 + \frac{1}{8} \cdot I^3$.

However, in the case of large volume visualization, we have to apply a multiresolution model, which introduces a certain overhead. The storage costs of multiresolution volume hierarchies are dominated by the cubic growth of the volumetric elements. A simple I^3 volume octree will introduce an overhead of $\frac{I^3-1}{7}$ and in total require $\frac{8 \cdot I^3-1}{7}$ elements, or $\approx \frac{8}{7} \cdot I^3$. In that context, the multiresolution costs for the two TA multiresolution models as presented in Sec. 4.1 are analyzed. The simplest multiresolution DVR data structure, as shown in Fig. 4.1(a), consists of all octree nodes being individual tensor decompositions. Such a model results in an octree hierarchy of x bricks B^3 represented by x core tensors $\mathscr{B}_{\text{brick}}$ and $3 \cdot x$ matrices $\mathbf{U}^{(n)}_{\text{brick}}$. In contrast, our second model (Fig. 4.1(b)) consists of a set of global mipmapped factor matrices $\mathbf{U}^{(n)}_{\downarrow k}$ and an octree hierarchy of core tensors $\mathscr{B}_{\text{brick}}$.

The matrices costs differ for the two multiresolution models. Mainly, the model with the global mipmapped factor matrices profits from reusing the matrices for all the bricks along a given row-selection of each factor matrix while the bricked TA model has to store three individual matrices for each octree node.

The size of one core tensor (i.e., one octree node) is only dependent on the rank R_n. The maximum rank in both multiresolution models is given as half of the brick size, i.e., $R_n = \frac{1}{2} \cdot B$. For the multiresolution models, we typically consider equally-sized subvolumes B^3 for octree nodes and hence we use also equal ranks along the three modes, i.e., $R = R_n \forall n$.

The factor matrix storage costs for the bricked TA model is given by three matrices of size $B \cdot R$ per octree node. The cumulated matrices of all leave nodes along one mode are of size $B \cdot R \cdot \frac{I^3}{B^3}$, where $R = \frac{1}{2} \cdot B$. That is in terms of I along all three modes: $\frac{3}{2 \cdot B} \cdot I^3$. Incorporating the general octree overhead, this results in a total factor matrices size of $\frac{8}{7} \cdot \frac{3}{2 \cdot B} \cdot I^3 = \frac{12}{7 \cdot B} \cdot I^3$. For the chosen brick size $B = 32$ this is $\frac{3}{56} \cdot I^3$.

The overall storage cost of the mipmapped factor matrices is approximately twice as large as the accumulated sum over all leaves brick factor matrices or the initial global factor matrices, due to the binary octree hierarchy over the factor matrix rows. Additionally, we store some information such as the core tensor rank value for each octree node in the global TA bases model, although this is neglected in this analysis. Intuitively, it can be seen that the matrices cover the full spatial data over all resolutions, that is $2 \cdot 3 \cdot I \cdot R$. However, the chosen brick size for the multiresolution model affects the initial rank size. The brick size in the global mipmapped factor matrix TA model was $B = 64$ and the initial rank was half the brick size, i.e., $R = \frac{1}{2} \cdot B$. Therefore, the matrix storage space required for the global TA bases model is $2 \cdot 3 \cdot I \cdot \frac{1}{2} \cdot 64 = 192 \cdot I$.

In both models the octree nodes store core tensors of size R^3. Since the ranks are chosen as half the initial mode size, we get an octree typical cost, but with half the dimension $\frac{1}{2} \cdot I$. This results in a total octree cost for all core tensors of $\approx \frac{8}{7} \frac{I^3}{2^3} = \frac{1}{7} \cdot I^3$.

Thus, we get a total storage cost of the bricked TA model and of the global TA bases model of $\frac{3}{56} \cdot I^3 + \frac{1}{7} \cdot I^3 = \frac{11}{56} \cdot I^3 \approx 0.20 \cdot I^3$ and $192 \cdot I + \frac{1}{7} \cdot I^3$, respectively. Since the dominating factor is the cubic term we conclude that our two TA multiresolution data structures require about eight times less than a normal volume octree, in terms of stored elements. In comparison, for large I the entire tensor decomposition cost ($\frac{3}{2} \cdot I^2 + \frac{1}{8} \cdot I^3$ from above) is less than $\frac{1}{56}$ larger but supports adaptive multiscale and multiresolution reconstruction and maintains much smaller core tensors. Note that typically, an octree consists of empty nodes, which are simply encoded with zeros. The theoretical costs mentioned above do not consider empty bricks since this is highly data-dependent.

Once the initial transform into a multiresolution model is performed, further data reduction can be achieved by quantizing the floating point TA coefficients. For this purpose a Tucker tensor-specific quantization approach was developed.

5.2 Tucker Tensor-specific Quantization

As mentioned previously, the core tensor and factor matrix coefficients take up unnecessary space if maintained as floating point values. For a compact representation of the tensor decomposition and to reduce the disk to host to device bandwidth during rendering, we apply a simple fixed bit length encoding based on tensor-specific quantization, as indicated in [Suter et al., 2011]. For example, a fixed bit length approach was selected in order to simplify parallel decoding on the GPU. In particular, the factor matrices and the core tensor of the Tucker model have a different distribution of coefficients and thus the quantization approach was selected accordingly.

5.2.1 Factor Matrices and Core Tensor Coefficients

The coefficients of the factor matrices $\mathbf{U}^{(1...3)}$ are normalized and distributed between $[-1,1]$, due to the orthonormality of factor matrices in the Tucker model. Therefore, a uniform linear 8-bit or 16-bit quantization as in Eq. 5.1 can effectively be applied. We use a single *min*/*max*-pair of the original data range to indicate the quantization range for all three factor matrices. In this way, the number of coefficients that need to be loaded for the reconstruction are minimized. The factor matrix values x are quantized to the values $\tilde{x}_{\mathbf{U}}$, where the range of the quantized coefficients with a bit-depth of $Q_{\mathbf{U}}$ is given by $(2^{Q_{\mathbf{U}}}-1)$.

$$\tilde{x}_{\mathbf{U}} = (2^{Q_{\mathbf{U}}}-1)\cdot\frac{x-x_{min}}{x_{max}-x_{min}} \tag{5.1}$$

As per definition of the Tucker model, the core tensor $\mathscr{B}$ captures the contribution of the linear bases combinations, i.e., the *energy* of the data, in its coefficients. The distribution of the signed coefficients is such that the first entry of the core tensor has an especially high absolute value close to the volume's norm, capturing most of the data energy, while many other entries concentrate around zero. The probability distribution of the other values between the two extrema is decreasing with their absolute magnitude in a logarithmic fashion (see Fig. 5.1). Hence we apply a logarithmic quantization scheme as in Eq. 5.2 for the core tensor coefficients, using a separate sign-bit. The absolute core tensor coefficient values $|x|$ are quantized to the values $|\tilde{x}_{\mathscr{B}}|$, where the range of the quantized coefficients with a bit-depth of $Q_{\mathscr{B}}$ is given by $(2^{Q_{\mathscr{B}}}-1)$.

$$|\tilde{x}_{\mathscr{B}}| = (2^{Q_{\mathscr{B}}}-1)\cdot\frac{\log_2(1+|x|)}{\log_2(1+|x_{max}|)} \tag{5.2}$$

Special treatment can be given to the one first high energy value mentioned before. It is known that this value, the *hot-corner coefficient*, is always at position

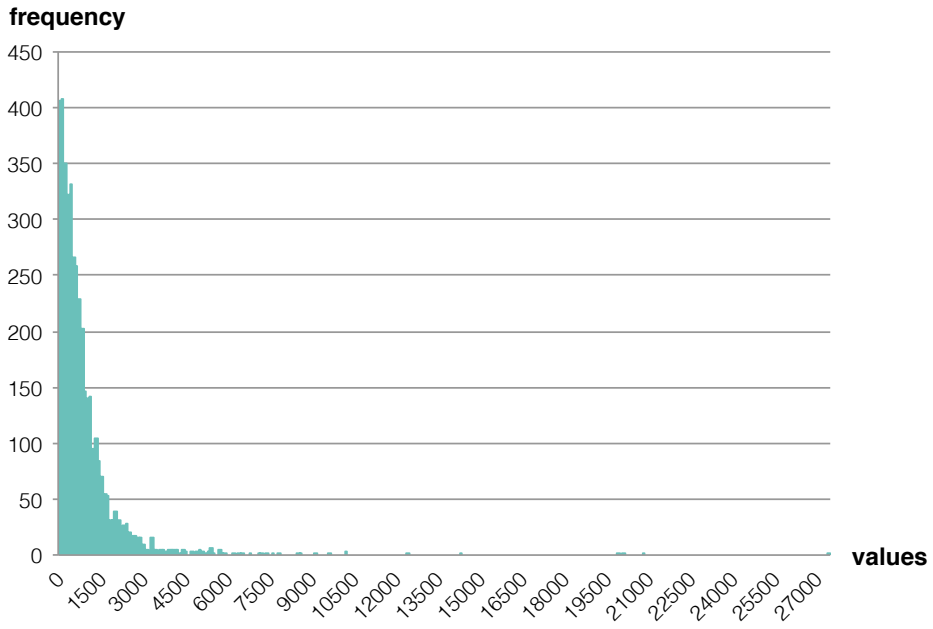


Figure 5.1: *Histogram of the absolute core tensor coefficient values. The core tensor coefficients are taken from a rank-*$(16,16,16)$*) TA on the hazelnut dataset. Most values are close to zero; only a few values reach up to the largest core tensor coefficient values. The by far largest core tensor value (*$119'894$*) is not even displayed.*

$\mathscr{B}(0,0,0)$. Since it is one value and in order to give more space for the quantization range to the other coefficients, we optionally fully encode the hot corner coefficient as a floating point value.

Various quantization levels for the other coefficients, $Q_{\mathbf{U}}$ and $Q_{\mathscr{B}}$, could be used and are thus analyzed. The quantization of the tensor coefficients helps to keep the critical CPU-to-GPU data transfer and disk storage low. In the next section, we analyze the error due to quantization and how the storage size is thus affected. We considered quantization approaches that use the same bit-length (from 8-bit to 16-bit) for all values within a coefficient type, the factor matrices $\mathbf{U}^{(\mathbf{1}\ldots\mathbf{3})}$ and the core tensor $\mathscr{B}$.

5.2.2 Storage Cost

The storage cost for different quantization approaches is indicated in Fig. 5.2, where U and B indicate factor matrices or core tensor settings, respectively, and *k*lin/log indicates linear or logarithmic quantization to $Q_{\mathbf{U},\mathscr{B}} = k$ bits according to Eqs. 5.1 and 5.2. The left-most value A:16 represents the size of a 2048^3 16-bit input volume dataset $\mathscr{A}$, and U:32 B:32 a 32-bit floating point representation of the reference rank-$(1024,1024,1024)$ reduced tensor approximation of $\widetilde{\mathscr{A}}$. The

data reduction follows the storage requirements outlined in Sec. 5.1.

We can see in Fig. 5.2 that the proposed quantization (U:8 B:8 to U:16 B:16) has an additional storage reduction effect, compared to the floating point tensor (U:32 B:32) and the original volume (A:16) data representation. Furthermore, for the quantized 32^3-bricked multiresolution octree hierarchy the storage consumption is minimally different from the non-bricked quantized format. Only the non-quantized bricked floating-point representation has an adverse space cost behavior due to its many coefficients that have to be stored. The approximation quality of the different quantization levels is analyzed next. From the storage cost results we can conclude that it is preferable to spend 16-bits on the factor matrix entries rather than on the core tensor, as the factor matrices $\mathbf{U}^{(1...3)}$, being quadratic, affect the total storage marginally compared to the core tensor $\mathscr{B}$, being cubic.

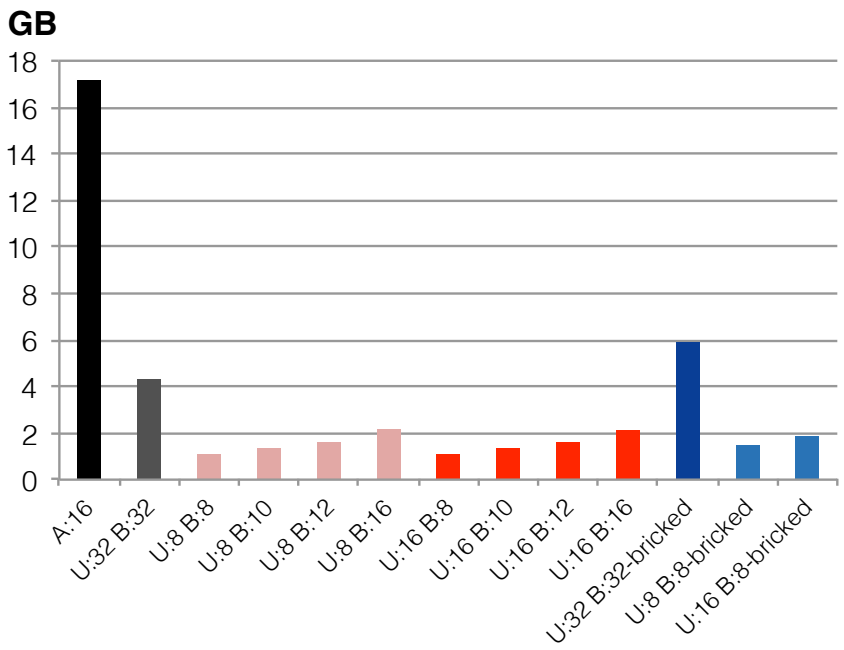


Figure 5.2: *Storage needed (in GB) for the various quantization approaches. U stands for the factor matrices, B for the core tensor. The number after B and U gives the number of bits used for the respective coefficient type. From [Suter et al., 2011].*

From Sec. 5.1, we know the overall storage costs of the bricked TA model and of the global TA bases model. In fact, actual storage costs are independent from the bit-depth of the initial dataset, e.g., 8-bit or 16-bit values. The storage costs are defined by the coefficients, resulting in an effective storage cost of $2 \cdot \frac{3}{56} \cdot I^3 + \frac{1}{7} \cdot I^3 = \frac{3}{28} \cdot I^3 + \frac{1}{7} \cdot I^3 = \frac{1}{4} \cdot I^3$ (bricked multiresolution TA) and $2 \cdot 192 \cdot I + \frac{1}{7} \cdot I^3 = 384 \cdot I + \frac{1}{7} \cdot I^3$ (multiresolution TA with global mipmapped factor matrices) for

16-bit factor matrix coefficients and 8-bit core tensor coefficients.

Finally, we strive to give some insight into the actual storage requirements (including empty space skipping and including the octree overhead) for our experiments. In the first multiresolution model (bricked multiresolution TA), the achieved storage costs for a 2048-cubed dataset (great ape molar) was 5.5GB. The difference between the actual 5.5GB and the theoretical 2.1GB ($0.25 \cdot 2048^3$) arises from the Berkeley DB overhead (see also Sec. 7.5.1). The $1024^2 \times 1080$ sized chameleon dataset was reduced to 230MB. However, the chameleon dataset involves a significant amount of empty space skipping. Without empty space skipping it should be about twice the data size, i.e., around 500MB. These storage cost requirements from the bricked TA model could be improved with the global mipmapped factor matrix multiresolution model. Namely, the 2048-cubed dataset could be reduced to 1.2GB and the chameleon dataset could be reduced to $162MB$ without empty space skipping.

5.2.3 Quantization Error

To evaluate the approximation quality of a rank-reduced and quantized tensor decomposition we use the signal-to-noise ratio (SNR) to express the error in relation to the data's signal strength. We define the signal strength of a volume $\mathscr{A}$ as the averaged Frobenius norm $\|\mathscr{A}\|_{\bar{F}} = \sqrt{\frac{1}{N}\sum a_{i_1,i_2,i_3}^2}$, and the approximation and quantization noise of the reconstructed volume $\widetilde{\mathscr{A}}$ as the root-mean-squared error (RMSE) $\varepsilon_{\widetilde{\mathscr{A}}} = \sqrt{\frac{1}{N}\sum(a_{i_1,i_2,i_3} - \tilde{a}_{i_1,i_2,i_3})^2}$. Hence the SNR is defined as $\sigma_{\widetilde{\mathscr{A}}} = 20 \cdot \log 10 \frac{\|\mathscr{A}\|_{\bar{F}}}{\varepsilon_{\widetilde{\mathscr{A}}}}$.

As base reference to evaluate quantization effects, we compare to the error which was introduced by a reduced rank-(R_1, R_2, R_3) tensor approximation $\widetilde{\mathscr{A}}$ of the original volume $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, where $R_n = \frac{1}{2} \cdot I$, as seen previously. In view of the processing time required, the costly approximation error analysis was not performed on a large volumes, but on three volume datasets of size $256^2 \times 128$. From the datasets described in Appendix A, the bonsai tree, the engine, and a subvolume of the great ape molar were chosen.

The triple-bars in Fig. 5.3 are organized in the indicated dataset order and bright-dark-medium-luminance color coded. The reference floating-point tensor decomposition (U:32 B:32) is shown to isolate and evaluate the quantization effect. We analyzed the quantization approaches outlined, applying linear and logarithmic quantization to both, the factor matrices and the core tensor. The effects on the approximation error were analyzed for entire as well as bricked volume Tucker decompositions. Fig. 5.3 shows the analysis of the approximation quality in terms of the SNR $\sigma_{\widetilde{\mathscr{A}}}$ for different linear and logarithmic quantizations.

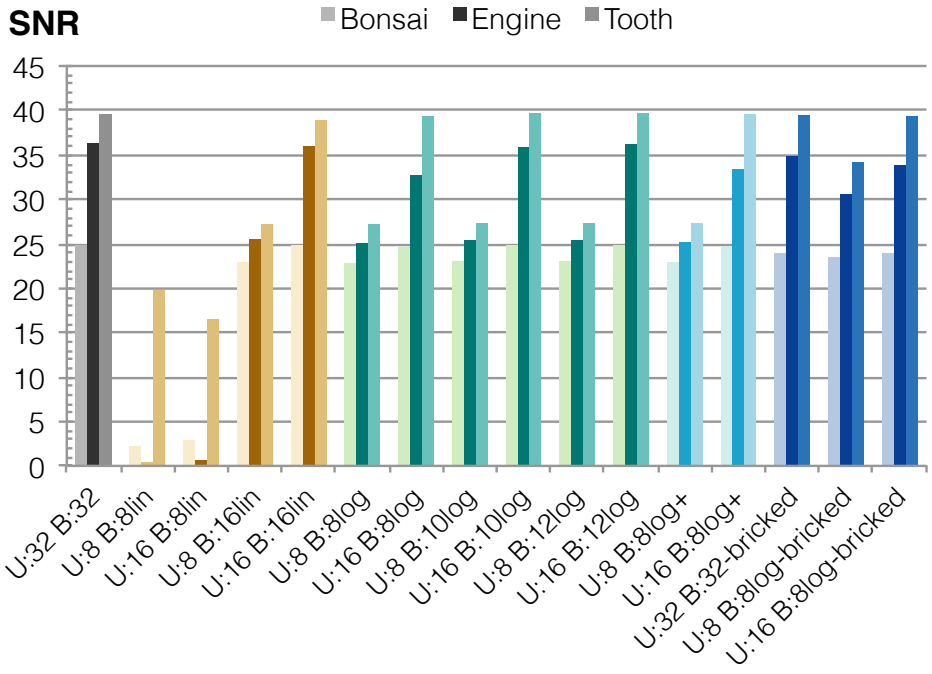


Figure 5.3: *Quantization error as SNR for various quantization approaches. The triple-bars are organized in the indicated dataset order and are bright-dark-medium-luminance color coded. U stands for the factor matrices, B for the core tensor. The number after B and U gives the number of bits used for the respective coefficient type. From [Suter et al., 2011].*

Except for the 8-bit linear core quantization (B:8lin), it is clear from Fig. 5.3 that for the factor matrices a significant improvement in SNR, and hence lower approximation error, can be achieved when using 16-bit (U:16) instead of 8-bit (U:8) quantization. Although the bonsai tree dataset does not benefit as strongly from this as the other volumes.

With respect to the core tensor quantization, it can be seen that the logarithmic is superior to the linear quantization, reaching comparable SNR values using much fewer bits, i.e., B:8log achieving almost the same quality as B:16lin for the same factor matrices quantization. It can be seen that increasing the quantization resolution from 8 to 12-bit only minimally improves the SNR, with the latter (B:12log) basically matching the more costly linear quantization. We evaluated the separate floating point representation of the hot-corner core tensor coefficient (B:8log+ in Fig. 5.3), which otherwise potentially wastes quantization resolution better spent on the remaining core tensor coefficients. This way the SNR can thus be slightly increased at the expense of only 4 extra bytes.

Taking the results from the storage cost study into account, the optimally com-

pact quantization can be achieved using a 16-bit linear factor matrix and a 8-bit logarithmic core tensor quantization with separate hot-corner (U:16 B:8log+).

In a bricked multiresolution octree setting the quantization quality differs only slightly, as shown in Fig. 5.3 (U:.. B:..-bricked), sometimes even being better. This could be explained by the fact that the bricked representation uses more coefficients in total over all bricks for the same volume dataset, consuming a little bit more space. The preferable optimal quantization setting is thus the same as above for the bricked TA, too.

5.3 Discussion

The tensor-specific quantization approach was applied to both multiresolution models. In the application with the bricked tensor decompositions the factor matrices and the core tensors were quantized with the suggested quantization scheme of 16-bit factor matrices and 8-bit core tensors. In the application with the global mipmapped factor matrices, only the core tensors were encoded, the matrices were fully loaded as floating point values for the whole application run-time. The difference between the storage costs two TA multiresolution models is seen in the costs of the factor matrix encoding, which results in a rough data reduction ratio of 0.15 and 0.25 of the original data size for the global factor matrices and the bricked TA matrices multiresolution models, respectively.

The final storage costs are independent from the bit-depth of the original volume and are mainly dominated by the cubic core tensors in the octree hierarchy. Following this analysis, the evaluated quantization scheme was a critical starting point to keep the storage cost low, namely by storing the core tensors with 8-bit values. This achievement was possible thanks to the observation that the core tensor values are logarithmically distributed and, therefore, a logarithmic (instead of a linear) quantization scheme saves a lot of precision for the final reconstruction. Until now, no other tensor-decomposition-specific quantization scheme as published in [Suter et al., 2011] was presented. The only other known tensor quantization scheme was presented in [Wu et al., 2008] who proposed a fixed linear quantization for the factor matrices (8-bit) and a variable quantization core tensor (8-20-bit). In [Suter et al., 2011], the goal was to refrain from variable-length coding to avoid the corresponding costly decompression.

The storage costs of the developed compression-domain multiresolution TA models achieve a well-comparable storage cost reduction compared to other approaches. In [Gobbetti et al., 2012], for example, a comparison between three different compression-domain DVR approaches indicates that the TA models are competing for the best storage costs. The 1024-cubed chameleon dataset, for example, was encoded for the highest resolution with 1GB for the K-SVD-based

dictionary, [Gobbetti et al., 2012], with 145MB for the hierarchical vector quantization, [Schneider and Westermann, 2003], and with 357MB for the brick-wise TA model, [Suter et al., 2011]. The corresponding storage cost for the global mipmapped TA bases is 162MB, which is close to the best performing vector quantization approach.

5.4 Summary

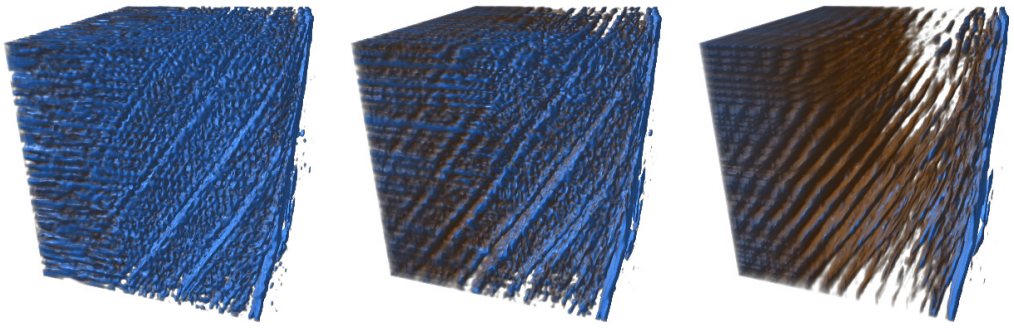
TA applied to multiresolution DVR reduced the amount of storage significantly. The storage costs are lowest for the global mipmapped TA model due to the reduced factor matrix storage costs (1GB instead of 5.5GB in the bricked TA model for an initial 2048-cubed dataset). In the global factor matrix TA model, the storage costs are dominated by core tensors of the octree hierarchy, while in the bricked TA model, the storage costs are around 25 percent of the original volume data size. Compared to other state-of-the-art and new compression-domain DVR systems, the storage cost reduction achieved is successful and effective (see Sec. 5.3).

Furthermore, a tensor-specific quantization scheme was developed during this thesis, fitting the value distributions in the factor matrices and the core tensors respectively. Namely, the orthonormal factor matrices were quantized in the bricked multiresolution model with a larger value range, 16-bit, and the core tensors were quantized by a logarithmic data distribution over a smaller data range, 8-bit.

The results and the storage cost analysis show that data reduction by TA is a competitive and effective approach that was introduced to compression-domain multiresolution volume visualization during this thesis. Besides the goal of achieving at a high compression ratio, there are certain quality requirements that need to be met by the reconstruction. In the next chapter, it is shown that we achieve not only a reduced storage cost, but a good reconstruction quality and and an approximation-quality feature-sensitive data visualization approach. Specifically, the TA approach helps to highlight relevant data features at different spatial scales. The background and experiments on this so-called multiscale feature detection by TA are the core topic of the next chapter.

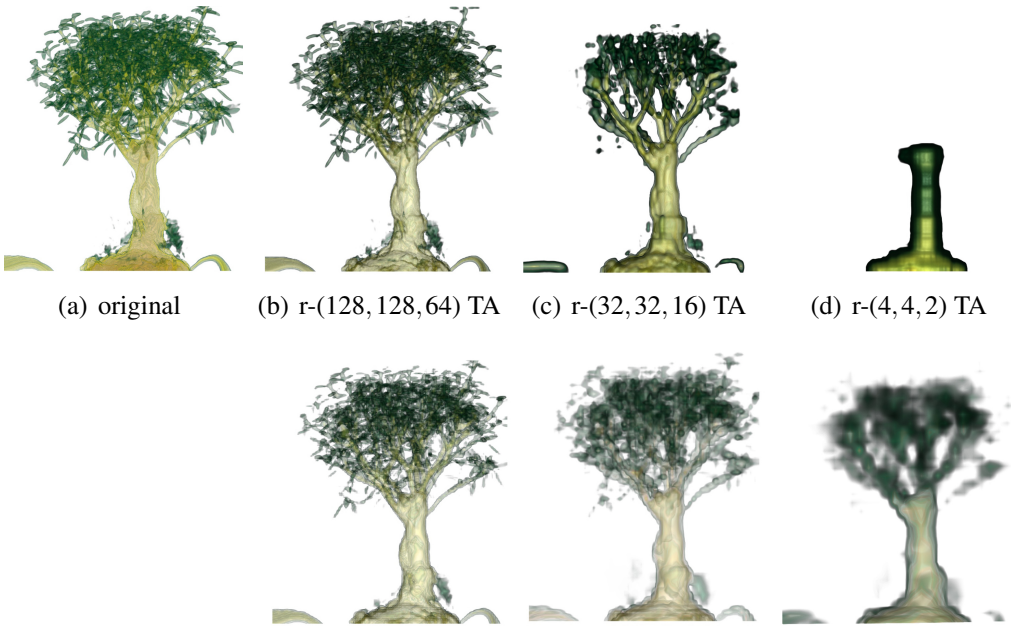
CHAPTER 6

RESULTS: MULTISCALE FEATURES IN VOLUME VISUALIZATION



6.1 Multiscale Volume Visualization

By multiscale volume visualization, we mean to have a parameter that is responsible for highlighting features (e.g., biological) at different spatial scales. We interpret the feature scale-space in a traditional way such that at coarser scales only the larger and more prominent structural components should be maintained as features, and more detailed features are identified on finer scales. In other words, the parameter to tune multiscalability corresponds to displaying an object at an abstract level (main shape of the data) or displaying the same object in details. This notion of multiscalability is different from the multiresolution concept, as multiscalability aims at extracting the most dominant features, while multiresolution is rather a spatial averaging of the whole dataset (see Fig. 6.1). Multiscalability is an approach known from PCA. Similarly, we exploit multiscalability for direct volume rendering from higher-order PCA, as previously introduced using tensor approximation (TA). In this context, the *scale* is given by the rank reduction (truncation) or number of coefficients used in the approximation. That is, the rank is the main parameter, which steers the display of features at multiple scales. Details on how the tensor rank truncation results in multiscalable direct volume visualization is shown in the next section.



(a) original (b) r-$(128,128,64)$ TA (c) r-$(32,32,16)$ TA (d) r-$(4,4,2)$ TA

(e) $2\times$ averaged (f) $4\times$ averaged (g) $8\times$ averaged

Figure 6.1: *(b-d) Idea of multiscalability (rank-(R_1,R_2,R_3) TA) vs. (e-g) idea of multiresolution (averaging) shown with the bonsai tree dataset (a) of size* $256^2 \times 128$.

6.1.1 Multiscalability within TA a Framework

In this thesis, we represent a volume dataset $\mathscr{A}$ spanned along the spatial axis x, y, z by a 3^{rd}-order tensor. This tensor is decomposed with so-called tensor decomposition approaches into a set of bases, consisting of three factor matrices $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, and $\mathbf{U}^{(3)}$ (one matrix per mode) and a 3^{rd}-order core tensor $\mathscr{B}$ with coefficients that describe the relationship between the original data and the factor matrices. One eminent feature of tensor decompositions is that we can apply a tensor rank truncation similar to a matrix rank truncation. The rank truncation is good for two things: first, we reduce the number of coefficients and ergo the amount of storage needed, and second, we use less detailed or redundant information from the dataset. Eventually, the data is stored and loaded as a truncated tensor decomposition. The original data is only reconstructed to its approximation $\widetilde{\mathscr{A}}$ when it is used for visualization.

The Tucker decomposition (one of the TA models) was chosen for compact data representation. As defined in Sec. 3.4.1, the Tucker model defines a rank-(R_1, R_2, R_3) approximation, where a small R_n corresponds to a low-rank approximation (many details removed) and a large R_n corresponds to an approximation more closely matching the original. In the Tucker model, the rank R_n for the initial decomposition has to be explicitly given. However, further adaptive rank truncations can be applied after the initial decomposition (similar to the rank truncation in the matrix SVD case). Even though the ordering of the coefficients in the core tensor is not strictly decreasing, as in the matrix SVD case, in practice it can be shown that progressive tensor rank truncation in the Tucker model works well for adaptive visualization of the data at different feature scales. Fig. 6.2 compares the progressive rank truncation from an initial rank-$(256, 256, 256)$ TA (bottom row) to a specific fixed rank-(R_1, R_2, R_3) tensor decomposition (top row) of a 512^3 input volume. Both reconstructions are visually similar down to the lowest ranks, which are rarely used, however. Furthermore, the round hazelnut shapes could be extracted well at lower ranks where higher ranks expose adding details. That is, the rank serves as a multiscale parameter in this example.

With the TA approach, the initial project idea (hypothesis) to use one common framework for visualization, data reduction and feature extraction is maintained. During two subprojects of this thesis [Suter et al., 2010a; Suter et al., 2011], it has been shown that the tensor rank parameter can be used to highlight features at different levels of scale. The applicability of TA for multiscale direct volume visualization is tested in the following section.

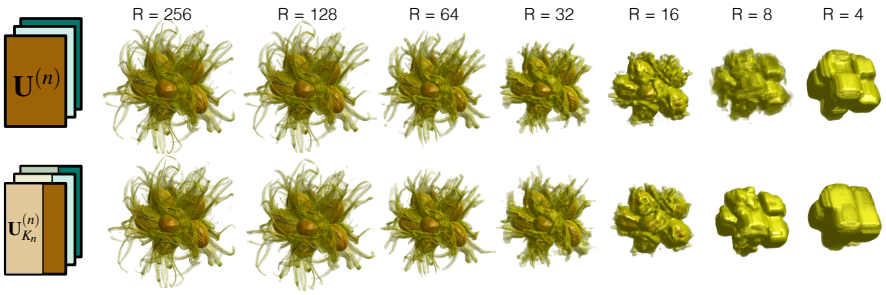


Figure 6.2: *Multiscale volume visualization by tensor rank reduction (bottom row) compared to rank-(R,R,R) tensor approximations to specific ranks R (top row), from [Suter et al., 2013].*

6.2 Application: Dental Microstructures

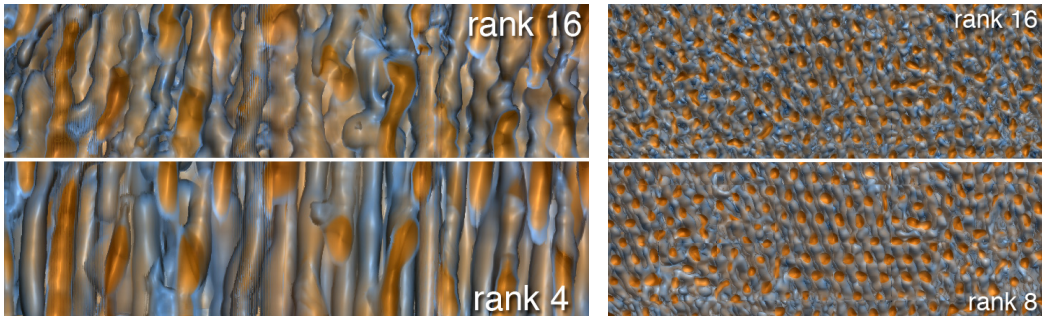
As outlined in Chap. 2, one goal of this thesis is to search for an approach to capture and identify the essential features of complex volumetric structures such as periodic growth patterns in tooth enamel scanned by μCT or PCST imaging (Sec. 2.1). With several experiments, different effects of rank-reduced tensor approximation and its applicability for multiscale volume visualization are tested. We start with a first experiment that shows the general feasibility of TA to enhance and highlight periodic features, then, we move on to a comprehensive comparison between the feature-preservability by TA and state-of-the-art wavelet transform (WT). Both, synthetic and real datasets were used. The results reported here are taken from [Suter et al., 2010b; Suter et al., 2010a; Suter et al., 2011].

6.2.1 Volume Features

Our structural volume features are defined by certain intensity regions, i.e., voxel elements (i,j,k) for which their value $\mathscr{A}[i,j,k]$ is in a given interval, in the volume dataset. The features we look at can manifest different characteristics at different scales. We interpret the feature scale-space in a traditional way: At coarser scales only the larger and more prominent structural components should be present as features, while at finer scales more detailed features should be identified. The scale in this context is given by the rank-(R_1,R_2,R_3) reduction or number of coefficients, used in the approximation $\widetilde{\mathscr{A}}$. Feature expressiveness is evaluated visually as well as numerically. Visually, the coarsening and structural simplification of features can be verified by comparing the display of the original dataset $\mathscr{A}$ to its approximation $\widetilde{\mathscr{A}}$. Numerically, we compare different approximations $\widetilde{\mathscr{A}}$ by their root-mean-squared error (RMSE) with respect to $\mathscr{A}$.

6.2.2 Highlight Features by TA

In an example taken from experiments with dental growth structures (see Sec. 2.2) as published in [Suter et al., 2011], we noticed that by using lower-rank TA, dental structures like growth prisms become highlighted, as illustrated in an example close-up in Fig. 6.3(a). A similar effect is shown in Fig. 6.3(b), where a horizontal cut orthogonal to the growth prisms (yellow dots) is visualized. The image of the base reference of a TA with tensor rank 16 shows the prisms' irregular spatial distribution. The lower-scale reconstruction with a tensor rank 8 clears out the fuzziness and reveals the layered periodic and parallel arrangement of the prisms.



(a) (b)

Figure 6.3: *(a) Dental growth structures (prisms), highlighted with a reduced tensor rank 4 reconstruction. Taken from a frontal projection to an area below the enamel surface (see Fig. A.2.7). (b) Dental growth structures (prisms), highlighted with reduced tensor rank 8 reconstruction. Taken from a horizontal cut through an area below the enamel surface (see Chap. 2) (from [Suter et al., 2011]).*

In order to determine the number of reduced ranks, we need to be aware of the original tensor rank. Results from different studies [Wu et al., 2008; Suter et al., 2010a; Suter et al., 2011] have shown that we can tolerate the error introduced by half of the original tensor rank, i.e., a half-ranked tensor decomposition is considered as the reference point. In Fig. 6.3 subvolumes of 32^3 are tensor encoded, i.e., we start with a tensor rank of 16 and a rank-(16,16,16) tensor approximation per brick.

6.2.3 Feature Expressiveness by TA and WT

To demonstrate the ability of TA to capture oriented patterns, tensor approximation and wavelet transform (WT), which is known as state-of-the-art method for multiresolution compact data decomposition, are applied to volume datasets containing multiscale features. Next, three experiments are conducted to test how TA

and WT compare in terms of feature expressiveness. In particular, axis-alignment of features and multiscalability are verified. The tests are first performed on synthetic, non-axis-aligned (dental) growth structures, then benchmarks are taken from two different real dental datasets.

The performance of TA and WT are compared at corresponding approximation levels. For each approximation, the same number of non-zero coefficients (NNC) has been chosen for both the WT and TA method. For TA, NNC is given by the rank reduction as $R_1 \cdot R_2 \cdot R_3 + R_1 \cdot I_1 + R_2 \cdot I_2 + R_3 \cdot I_3$, representing the number of coefficients in the tensor decomposition[1]. Correspondingly, in the case of WT, we selected the most significant wavelets' NNCs. Unless specified otherwise, we used the multilevel biorthogonal 9/7 wavelet transform, which is used in JPEG-2000 and in [Wu et al., 2008], too.

In Fig. 6.4, a synthetic dataset after [Macho et al., 2003] (see Fig. 2.1(b)) was produced in order to demonstrate the visual results of applying different compression ratios (columns in Fig. 6.4) with TA and wavelets (rows in Fig. 6.4). Besides TA, we observed that only the Haar wavelet managed to reconstruct the original structures, but only with a "high" NNC. With TA, we were even able to recognize the main directions of the original structures with low-rank approximations and ergo with only a few coefficients. In particular, we observed that TA was in favor when dealing with non-axis-aligned structures such as the curved fibers. The same effect can be observed for real 3D dental enamel growth patterns.

Fig. 6.5 demonstrates that a compact TA (rank-$(8,8,8)$ with $2'048$ coefficients) makes it possible to highlight features (growth prisms) that are difficult to identify and visualize in the original dataset, or on finer approximation scales. At a corresponding number of coefficients, the Haar or biorthogonal 9/7 WT approaches show difficulties in reconstructing the characteristic features. However, it could be argued that the features from Fig. 6.5 are axis-aligned. Therefore, another experiment is performed with non-axis-aligned features.

Fig. 6.6 visualizes different non-axis-aligned 3D dental enamel growth patterns. The periodic halts along growth prisms cause the formation of surface layers, which can be identified and visualized using TA at different scales using progressive rank-reduction. In particular, the weekly growth markers (Retzius lines) can be well analyzed using 3D visualization and variation of reconstruction scale. The biorthogonal 9/7 WT based approach fails to extract the growth-halt layers and continuously transforms into a blobby reconstruction at progressively reduced approximation level. Compared to Fig. 6.3 where TA is applied to subvolumes, in Fig. 6.6, the full dataset of size 256^3 was tensor encoded, i.e., we started with a tensor rank of 128.

[1] By default, the TA coefficients do not equal zero, neither a further coefficient thresholding was applied. TA coefficient thresholding needs further investigations and is a topic for future research.

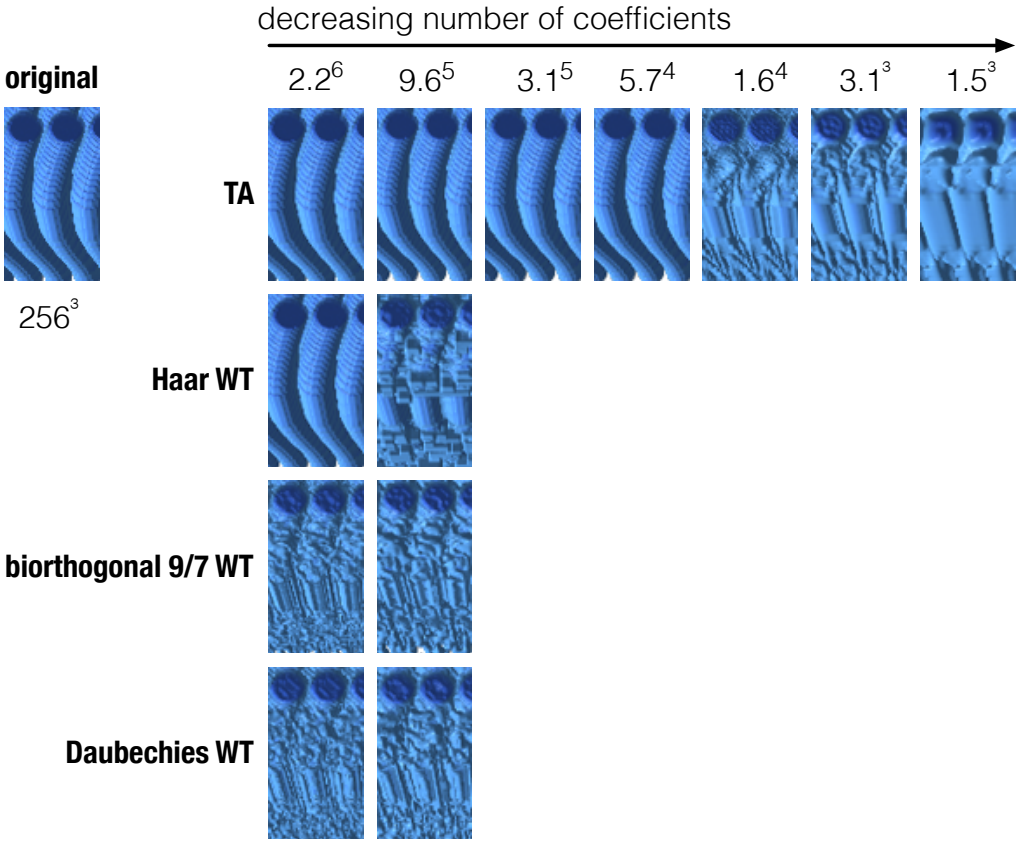
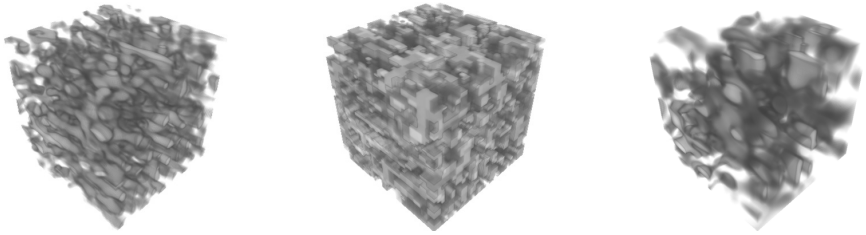
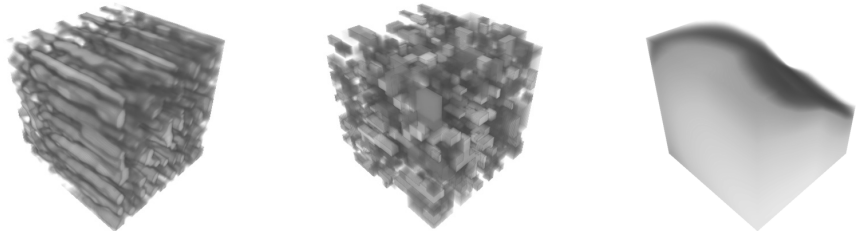


Figure 6.4: *Synthetic growth structures reconstructed with different numbers of encoded coefficients and different approximation approaches (adapted from [Suter et al., 2010a]).*

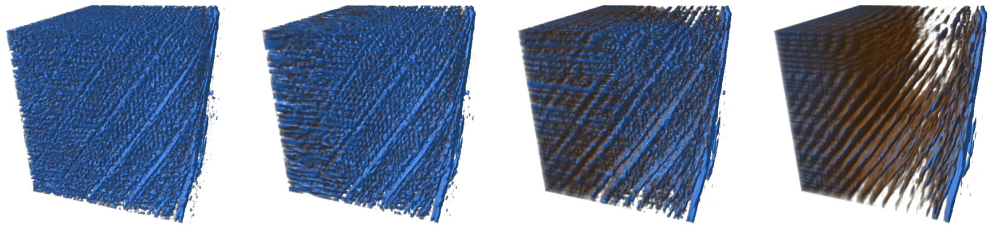


(a) Rank-(16,16,16) TA (b) 7'168 Haar WCs (c) 7'168 bior. WCs

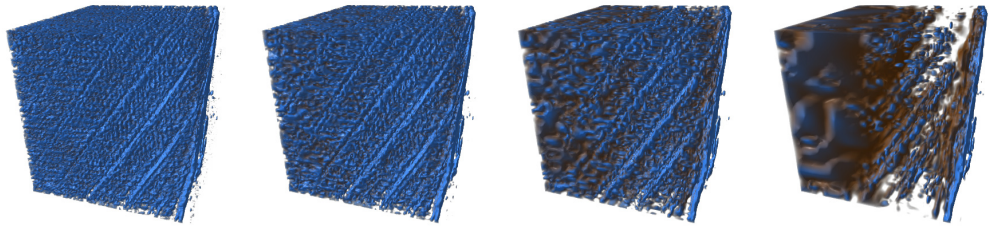


(d) Rank-(8,8,8) TA (e) 2'048 Haar WCs (f) 2'048 bior. WCs

Figure 6.5: *Structural volume dataset of tooth enamel acquired with PCST (*64^3 *voxels, 16-bit voxel depth, 0.75 microns resolution per voxel). Reconstructions from three different approximation levels: (a,d) TAs; (b,e) Haar WT, and (c,f) biorthogonal 9/7 WT.*



(a) tensor rank 128 (b) tensor rank 64 (c) tensor rank 32 (d) tensor rank 16



(e) 2^6 WCs (f) 3^5 WCs (g) 5.7^4 WCs (h) 1.6^4 WCs

Figure 6.6: *Periodic microstructures in tooth enamel (*256^3*, 16-bit voxel depth, 0.75 microns per voxel). (a-d) Feature visualization using different tensor rank approximations. Showing on the front side, horizontal growth prisms oriented left-to-right and diagonal Retzius lines oriented bottom-left to top-right. (e-n) Reconstructions from corresponding numbers of biorthogonal 9/7 wavelet coefficients (from [Suter et al., 2010a]).*

6.2.4 Rate Distortion (Numerical Approximation Quality)

Above, a qualitative evaluation of the feature-preserving reconstruction performance of TA in comparison to state-of-the-art WT based approaches was presented. In addition to the visual assessment of feature expressiveness, a quantitative numerical approximation analysis is required to fully establish the capability and potential of the proposed TA based feature extraction and visualization framework. Our numerical evaluation analyzes the performance of TA versus WT in terms of their rate-distortion. A rate-distortion diagram displays a curve between two axes (error and data approximation level), where a curve is the better the more closely it is aligned to the axes. We measured the root-mean-squared error (RMSE) ε over all voxels in the original and the approximated datasets and put ε in relation to the number of (non-zero) coefficients NNC used for different approximation levels. Note that we are interested in the feature difference at a certain scale and thus the RMSE $\varepsilon = \sqrt{m^{-1}\sum_{i,j,k}(\mathscr{A}[i,j,k] - \widetilde{\mathscr{A}}[i,j,k])^2}$ is computed only for the m voxels where $\widetilde{\mathscr{A}}[i,j,k]$ is within a given intensity range of interest (e.g., of the feature).

As can be seen in Fig. 6.7, for the real microstructure volume, the rate-distortion curves of TA and WT are close, with a slight advantage for the WT. The TA curve for the synthetic microstructure volume in terms of rate distortion, is less qualified than the WT curve. However, the visual images have shown a different result. We have observed that standard measures of data reduction quality, such as rate-distortion on RMSE, do not sufficiently capture the feature expressiveness of a numerical approximation method.

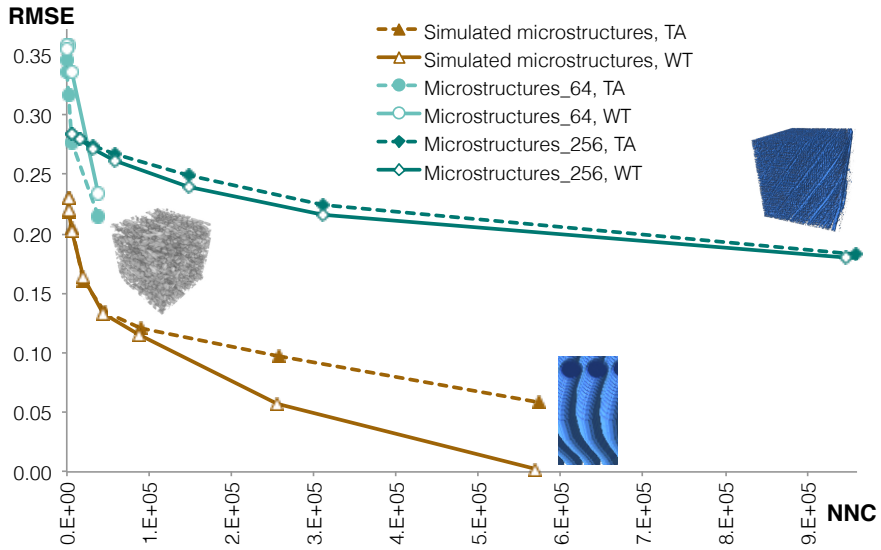


Figure 6.7: *Rate-distortion curves for tensor and wavelet reconstructions for datasets from Figs. 6.4– 6.6 (adapted from [Suter et al., 2010a]).*

6.3 Multiscalability and Multiresolution in One

As we have seen previously in this chapter, multiscale feature visualization is possible through tensor rank truncation. The first goal of this section is to integrate the multiscalability into a multiresolution model with the global TA factor matrices presented in Chap. 4. The second goal of this section is to develop a feature scale parameter, which in the end can be used in the visualization system to steer both, multiresolution DVR and multiscale feature visualization in one.

The key challenge to integrate the multiscalability is to maintain the tensor rank truncation. The tensor rank truncation works as long as we maintain the so-called *all-orthogonality property* within the core tensors. The all-orthogonality (see [De Lathauwer et al., 2000a]) is achieved when a core tensor is generated from orthogonal matrices. The TA factor matrices produced from a standard HOSVD procedure fulfill this property; in fact, the TA factor matrices are even orthonormal. However, in order to model multiresolution, we use spatial selection and spatial averaging in our global TA factor matrices. Hence, we can not produce all-orthogonal core tensors from those mipmapped global TA factor matrices.

We solved the issue of maintaining all-orthogonal core-tensors in [Suter et al., 2013]. The key idea was to apply SVDs on TA factor matrix row blocks corresponding to the octree bricks. That means, we re-span the subspace of each row-block global factor submatrix $\mathbf{U}_{J_n}^{(n)}$ by applying another SVD (similar to [Tsai and Shih, 2012]). As shown in Fig. 6.8, the row-block matrices' columns corresponding to bricks are then replaced with the orthogonal singular vectors. In that way, we are able to define per-brick core tensors that can be truncated. Intuitively, this recomposition of the global matrices can be seen as a different representation of the same local subspaces as defined by the initial non-orthogonal submatrices. J_n corresponds to the octree brick size (including borders). Due to equally sized bricks along all spatial octree directions, the sub-block replacements can be used for spatially-corresponding bricks.

For the multiscale feature scale manipulation within the visualization system, we compute a feature scale parameter for different rank truncated tensor reconstructions of every brick. Specifically, the feature scale parameter is computed on the differences of the approximations and the original at different feature scales (i.e., different rank truncations). Therefore, we compute per brick (excluding borders) the differences in terms of the root-mean-square error (RMSE). With respect to different resolution approximations, we use trilinear interpolation to compute the RMSE between any LOD brick and the original. We compute for every brick a number of different rank truncated reconstructions and store this information in a separate file.

Fig. 6.9 visualizes the average errors per LOD. The chart shows that the error

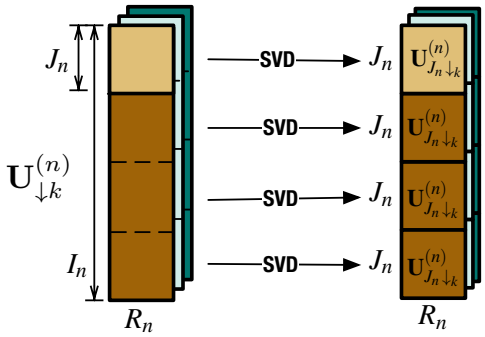


Figure 6.8: *Preprocessing the mipmapped initial global factor matrices in order to obtain orthogonal localized row-block submatrices and thus all-orthogonal per-brick core tensors in the octree hierarchy. From [Suter et al., 2013].*

is gradually decreasing when refining the resolution, and it overlaps between octree levels and rank ranges. Still, there is an overlap between the LOD RMSES of different octree levels that needs to be handled in the visualization algorithm. Nevertheless, the presented LOD RMSE was chosen to steer multiple features scales (chosen ranks dependent on brick LOD RMSE) and multiple resolutions (chosen brick LOD) during interactive visualization. As the truncated approximations of the Tucker model do not guarantee a strictly decreasing error, we map the minimum-maximum error range to the range of ranks $R_i = \{8, 9, 10, \ldots, 31, 32\}$.

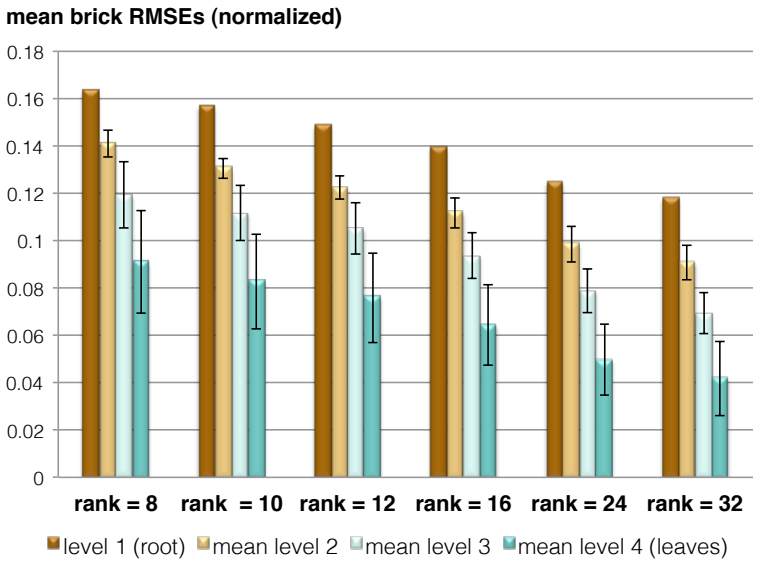
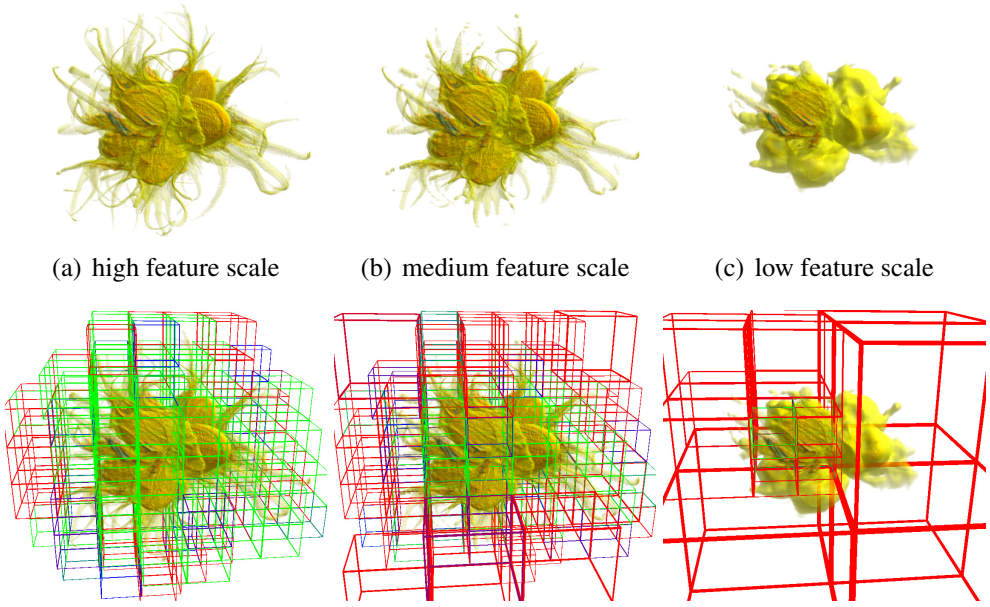


Figure 6.9: *Mean normalized brick RMSEs for all octree levels of the hazelnut. The standard deviation of the errors is additionally indicated. From [Suter et al., 2013].*

In practice, as can be seen in Fig. 6.10, the coupling of multiresolution DVR and multiscale feature visualization through the feature scale parameter works well. The resolution is indicated by the brick bounding box size and the rank by the color of the brick bounding box (red–blue–green for few–many–more ranks). In Fig. 6.10(a), for instance, we visualized a high feature scale with chosen high ranks and high resolutions. By decreasing the target feature scale, the chosen resolutions become coarser and lower ranks are chosen (Fig. 6.10(b)) until we reach a low feature scale, as shown in Fig. 6.10(c), where we see that only the finest details are encoded with high ranks.



(a) high feature scale (b) medium feature scale (c) low feature scale

Figure 6.10: *Coupling of multiresolution and multiscalability by a feature scale parameter (rank-based). The rank is color encoded (red–blue–green bricks correspond to few-more-many ranks); the size of each brick indicates its spatial resolution.*

6.4 Discussion

The results in this chapter show, first, the feasibility of the TA framework to produce multiscale feature visualization with DVR and, second, that the multiscale visualization can be coupled with the previously introduced multiresolution DVR. The most important insights are discussed here.

In [Wang et al., 2005b; Wu et al., 2008], it has been shown that TA can generate higher quality images at larger data reduction ratios than WT or PCA. In our work, we went one step further and we have shown that the mathematical framework of TA makes it possible to highlight features, which were difficult to see

in the original dataset (Figs. 6.6(b) and 6.6(c)). This is particulary applicable for features at multiple scales, which can be brought out with TA at corresponding approximation levels (Fig. 6.6). Notably at low ranks, i.e., at high data reduction ratios, TA showed higher quality reconstructions of internal structures compared to WT. While WT showed reconstructions with a most closely visual resemblance to the overall original appearance (multiresolution-fashion), TA identifies specific structural features at different scales (e.g., Figs. 6.6(b) or 6.6(d)).

Wavelets focus on optimal data reduction over the complete volume. That is, WT is beneficial when the overall statistical distribution of the dataset is intended to be reconstructed with a coarser resolution. In contrast, we follow an approach that extracts specific features based on statistical properties like the major direction or a periodicity. TA, similar to PCA, which extracts the major direction of a dataset, is more powerful regarding this latter aspect since it finds appropriate bases for reconstruction rather than assuming fixed basis functions as the WT. An approach like TA extracts components with more importance and neglects irrelevant areas within the dataset. Hence TA has the advantage when we want to analyze features, e.g., count the number of Retzius lines (Fig. 6.6(d)).

Compared to wavelets, where different wavelets need to be evaluated in order to find out which wavelet best fits a dataset, with the TA approach, we do not have to take such a decision with the TA approach, as there is a single mathematical tool, a rank-(R_1, R_2, R_3) TA. However, higher computational costs need to be considered for TA (learn basis with ALS algorithms), especially for large datasets, for which we need to consider bricking or similar data decomposition steps during volume processing and rendering.

Furthermore, we have observed that standard measures of data reduction quality, such as rate-distortion on RMSE, do not sufficiently capture the feature expressiveness of a numerical approximation method. New procedures need to be developed, which permit the quantitative analysis of feature selectivity in lossy data reconstruction methods. Examples of future investigations are, e.g., [Wu et al., 2010; Lin and Kuo, 2011; Hill et al., 2011].

The coupling of the multiresolution DVR and the multiscale feature visualization is completely based on the inherent properties given from the TA framework in the TA bases. Moreover, we could produce all-orthogonal rank-reducible per-brick core tensors for the multiresolution TA hierarchy from the global TA bases by applying row-block SVDs as outlined previously. We are not aware of any other system exploiting a similar visualization concept to that presented in this thesis and in [Suter et al., 2013]. Thus, in the future, it is suggested that more investigations into feature scale manipulation applications as well as in-depth experiments should be carried out.

6.5 Summary

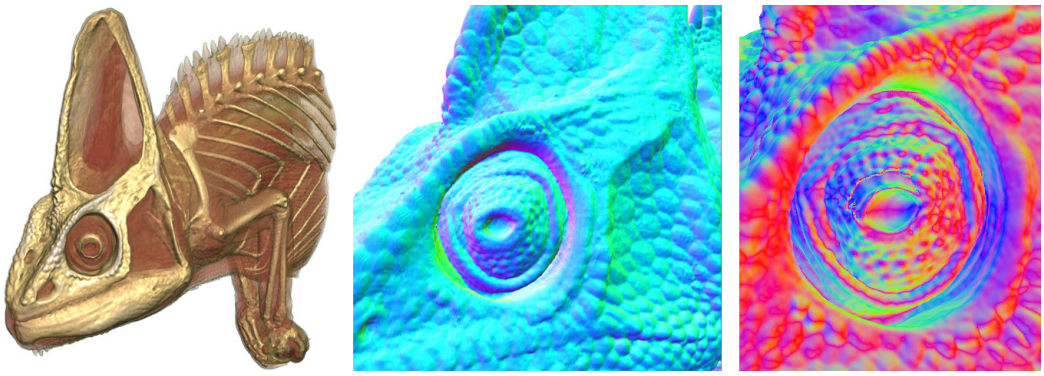
In summary, TA is powerful when we are not interested in the complete appearance of a dataset, but rather want to highlight or count features at a certain scale. Since new data acquisition techniques lead to volume datasets of ever-increasing size, which tend to be one step ahead of the available graphics resources for interactive visualization, there is thus an ongoing need to develop new data reduction and feature extraction methods to tackle the resulting performance bottlenecks. This thesis demonstrates that TA is a powerful approach to (a) represent microstructural volume datasets at high data reduction ratios, and (b) simultaneously highlight relevant features at different spatial scales.

On top of this unique chosen TA framework, we presented a new concept to couple multiresolution DVR and multiscale feature visualization within an unified framework. The idea exploits a novel TA hierarchy for both, multiresolution modeling and multiscale feature representation, and is implemented using a state-of-the-art GPU-based ray-caster. The multiresolution and multiscale TA properties are coupled through a feature scale parameter that can be operated at runtime by the user. The feature scale parameter is precomputed based on per-brick RMSE errors over a range of truncated approximations. By adjusting the feature scale parameter, the DVR implementation automatically chooses whether to increase or reduce spatial resolutions and feature scales.

The implementation details of all the presented multiresolution and multiscale TA DVR approaches are given in the next chapter.

CHAPTER 7

RESULTS: IMPLEMENTATION



7.1 TA Rendering Pipeline

The general TA rendering pipeline was already outlined in the introduction in Fig. 1.2. The focus of this chapter is the implementation of the newly proposed TA stages in the rendering pipeline (see Fig. 7.1). First, the brick compression or compact data representation using TA is explained, second, the construction of the multiresolution TA data structure is presented, third, the developed reconstruction strategy from the produced multiresolution data structure is shown, and fourth, the integration of TA into state-of-the-art multiresolution DVR visualization systems and its interactive performance are evaluated.

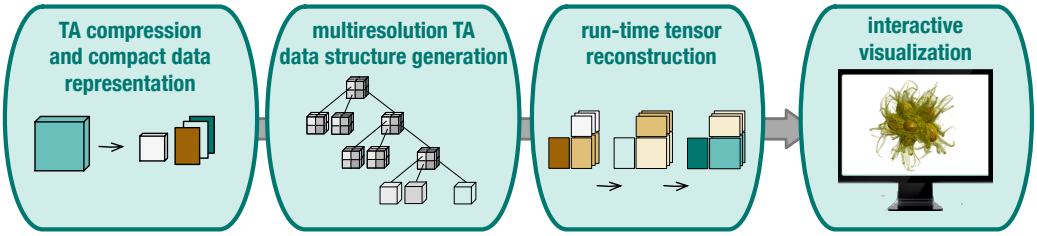


Figure 7.1: *The implementation pipeline of TA multiresolution and multiscale DVR.*

Regarding the implementation, different focuses were set during the development of the four stages. While the initial tensor decomposition of a large volume data is memory intense, the tensor reconstruction needs to be performed at run-time since interactive visualization is one of the predefined goals. The first two stages are carried out in a pre-processing routine and are, therefore, not run-time critical. However, it was needed to implement a memory-optimized tensor decomposition algorithm in order to decompose large volumes. The multiresolution data structure generation was neither memory-critical (small bricks are processed) nor run-time critical. The tensor reconstruction is mainly run-time critical, however, since it is computed on the GPU, it is to some degrees also memory-intense since GPU memory is limited. The same applies to the interactive visualization: Due to the brick-wise data loading the memory bottleneck is reduced, however, the total amount of currently loaded bricks on the GPU is limited. That is, even though the most recently used bricks are cached on the GPU, new data needs to be continuously transferred from CPU to GPU during interactive visualization. Therefore, the bandwidth plays another important role, which can, however, be reduced by achieving high brick compression ratios. The performance of the rendering is in fact another critical implementation stage. However, this issue was not addressed during this thesis. The memory-intense, run-time critical and bandwidth critical implementation stages are summarized in Tab. 7.1. The four implementation stages are elaborated and evaluated in detail in the next four sections.

	tensor decomposition	TA octree generation	tensor reconstruction	interactive visualization
memory-intense	✓✓✓		✓	✓
run-time critical			✓✓✓	✓✓✓
bandwidth critical			✓✓	✓✓

Table 7.1: *Memory-intense, run-time critical and bandwidth critical implementation stages. The number of checkmarks indicates the criticalness of an implementation stage.*

7.2 Tensor Decomposition Implementation

During this thesis project, the underlying data structures and algorithms used for each 3^{rd}-order tensor (tensor3) and the tensor decompositions were implemented in an open-source C++ library. The existing vector and matrix math library (vmmlib) (see [vmm, 2013]) was extended for 3^{rd}-order tensors. Specifically, data access to 3^{rd}-order tensors and different algorithms to perform the tensor decomposition were implemented. To our knowledge, this is the first commonly available tensor approximation library in C++. The well-established MATLAB toolbox by Bader and Kolda [Bader and Kolda, 2006] served as a reference.

7.2.1 CPU Tensor Classes

As stated before, the basic tensor data structure as well as the tensor decomposition and tensor reconstruction algorithms were developed and integrated into the template-based *vmmlib*. There is an implementation of a 3^{rd}-order tensor (tensor3), which is the main data structure containing frontal matrix slices stored in an array. For memory-mapping purposes a tensor memory mapper class was developed. In addition, there are the two main tensor models available as a class, the tucker3 tensor and the cp3 tensor. The tucker3 tensor uses the tensor3 HOOI implementation, while the cp3 tensor uses the tensor3 HOPM implementation. Both alternating least-squares (ALS) implementations (HOOI and HOPM) use the HOSVD, the HOEIGS and the tensor times matrix (TTM) implementations. There are further utilities for conversions and import/export available. The relevant tensor classes and their description are illustrated in a simplified UML class diagram in Fig. 7.2.

7.2.2 Initial Large Tensor Decomposition

In order to perform an initial decomposition on a large tensor input, some optimizations of the presented tensor classes had to be developed. As visualized

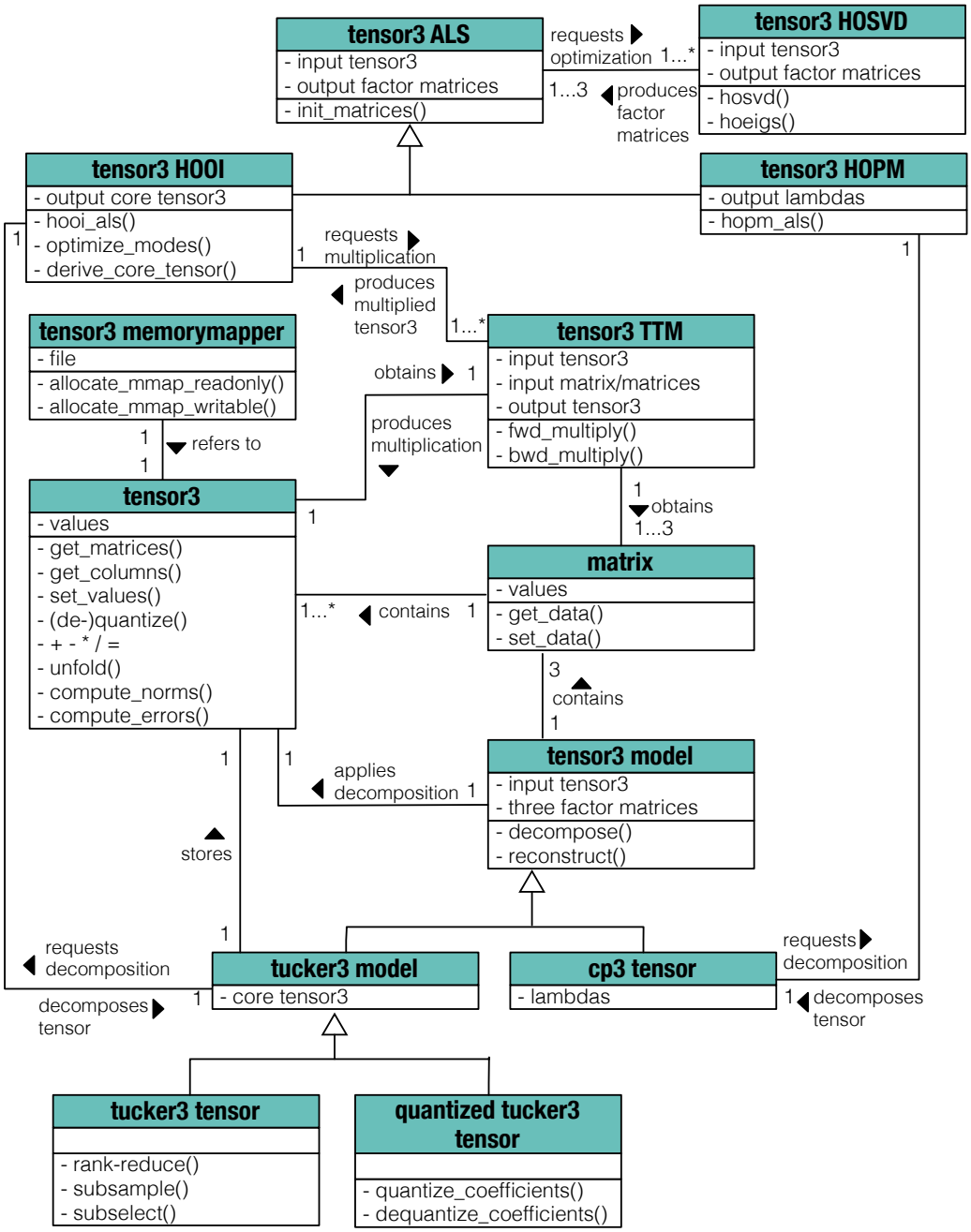


Figure 7.2: *Simplified class diagram of CPU tensor classes integrated into vmmlib.*

in Fig. 7.3, tThe actual method is based on an ALS algorithm implemented as a higher-order orthogonal iteration (HOOI) (see App. D). One iteration of the HOOI ALS for a 3^{rd}-order tensor (a volume) consists of three optimization steps, one along each mode. That is for one optimization along mode n, the data tensor is projected onto the other two factor matrices followed by extracting a new factor matrix for the optimized mode. As shown in Alg. 2 and Fig. 7.4, thus, two factor matrices $\mathbf{U}^{(n\pm1)}$ are fixed to optimize $\mathbf{U}^{(n)}$ in one iteration.

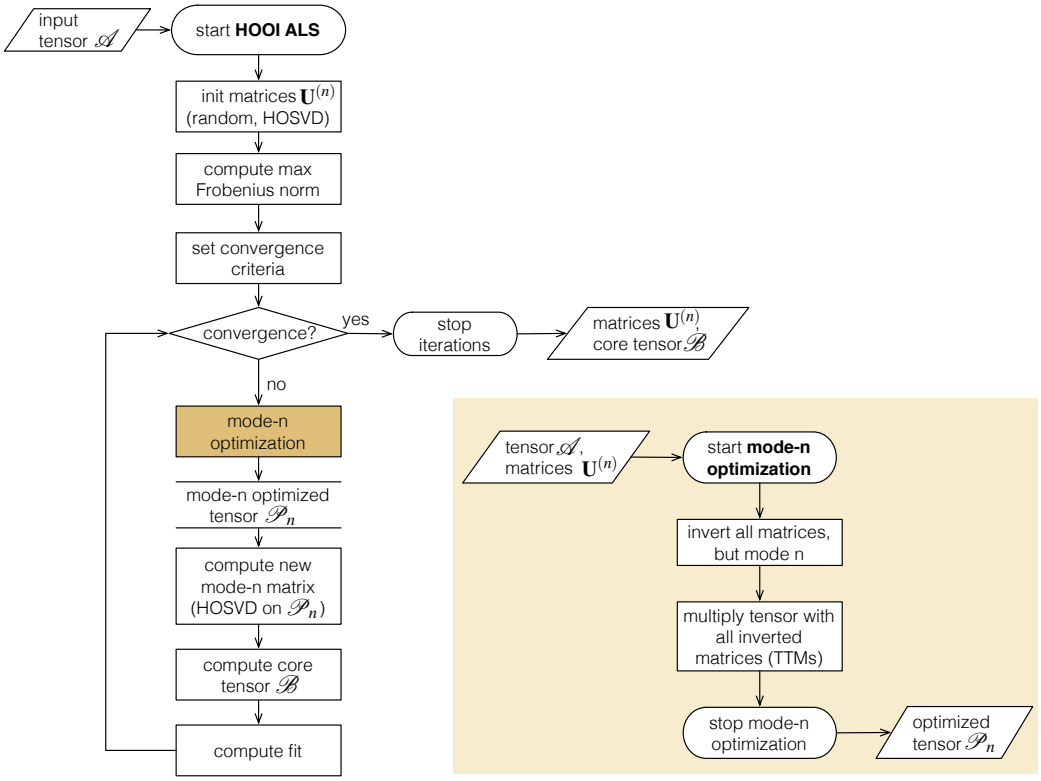
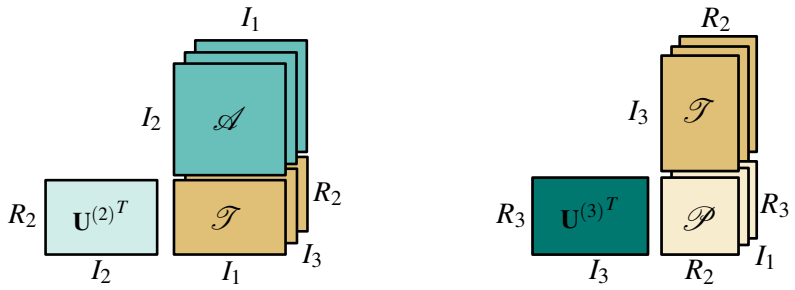


Figure 7.3: *Visualization of the HOOI ALS and the mode-n optimization.*

Algorithm 2 HOOI optimization of one mode, here mode $n = 1$, (see Alg. 16, lines 9…12).

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, R_n
2: **Output:** optimized $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$
3: **for** mode n optimization **do**
4: TTM: tensor $\mathscr{A}_{I_1 \times I_2 \times I_3}$ times matrix $\mathbf{U}^{(2)}_{I_2 \times R_2}{}^{T}$ multiplication $\rightarrow \mathscr{T}_{I_1 \times R_2 \times I_3}$
5: TTM: tensor $\mathscr{T}_{I_1 \times R_2 \times I_3}$ times matrix $\mathbf{U}^{(3)}_{I_3 \times R_3}{}^{T} \rightarrow \mathscr{P}_{I_1 \times R_2 \times R_3}$
6: HOSVD (Alg. 1) on $\mathbf{P}_{(n)}$ (unfolded $\mathscr{P}_n$ along mode n) $\rightarrow \mathbf{U}^{(n)}$
7: **end for**



(a) Alg. 2, line 4 (b) Alg. 2, line 5

Figure 7.4: *TTM multiplications needed in order to produce the projection tensor $\mathscr{P}$: n-mode products along all modes but mode n are performed.*

The HOOI decomposition of large datasets exhibits two main bottlenecks [Suter et al., 2013]: (1) the tensor times matrix multiplications (TTMs) of large tensors as in line 2 of Alg. 2, and (2) the HOSVD as in line 4 of Alg. 2. The two bottlenecks were addressed as follows:

1. Larger than main memory data tensors $\mathscr{A}$ are unfolded in the required mode direction ($\rightarrow \mathbf{A}_{(n)}$) once and accessed as memory mapped files.

2. The HOOI iterations are designed such that the TTM operations access the unfoldings of $\mathbf{A}_{(n)}$ sequentially and memory aligned.

3. The TTM operation has been implemented using parallel multi-threaded matrix matrix multiplications.

4. The HOSVD is computed either based on the SVD or the EIG, depending on the size of the input tensor.

7.2.3 Performed Optimizations

As evaluated previously, the HOSVD and the TTMs are the most critical computation steps. While the HOSVD is computing-intense, the TTMs are memory-intense. In particular, the first TTM of the first mode optimization (Alg. 2, line 2, and Fig. 3.14(a)) is the most critical one. The size of this first unfolded tensor is $I \cdot I \cdot I$, while all the successive TTMs are only applied to tensors of size $I \cdot I \cdot R$, $I \cdot R \cdot R$ or $R \cdot R \cdot R$, where $R \ll I$. The algorithms used for the large tensor decomposition have been integrated into *vmmlib*, for which wrappers to BLAS and LAPACK routines were developed during this thesis.

Optimized TTMs

The TTMs were optimized using BLAS DGEMM for all matrix matrix multiplications. For this, a TTM is implemented as multiple matrix matrix multiplications, where one matrix stays always the TA factor matrix and the other matrix is updated by the next tensor slice along the third axis. In order to parallelize this step, OpenMP was used. The underlying array of the tensor3 structure in *vmmlib* is implemented as several column-major frontal slices. In order to have a better access pattern within the array during the TTM, the TTMs were implemented with either forward (FWD) or backward (BWD) cyclic TTMs (see Fig. 3.11) depending on the access pattern during the TTMs. The specific memory-access optimized implementations for the three HOOI optimization steps as well as the core tensor computations are shown in the Algs. 3-7 (see also Alg. 16, lines 10 and 13).

Algorithm 3 Optimize mode $n = 1$.

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, $\mathbf{U}^{(\mathbf{2})}$, $\mathbf{U}^{(\mathbf{3})}$
2: **Output:** $\mathscr{P}_1 \in \mathbb{R}^{I_1 \times R_2 \times R_3}$
3: compute ${\mathbf{U}^{(\mathbf{2})}}^{-1}$ (use transpose for orthogonal matrices)
4: compute ${\mathbf{U}^{(\mathbf{3})}}^{-1}$ (use transpose for orthogonal matrices)
5: memory optimized version (after [De Lathauwer et al., 2000b]):
6: TTM (frontal BWD) multiplication of ${\mathbf{U}^{(\mathbf{2})}}^{-1}$ with $\mathscr{A} \rightarrow \mathscr{T}$ (Alg. 2, line 4)
7: TTM (horizontal BWD) multiplication of ${\mathbf{U}^{(\mathbf{3})}}^{-1}$ with $\mathscr{T} \rightarrow \mathscr{P}_1$ (Alg. 2, line 5)

Algorithm 4 Optimize mode $n = 2$.

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, $\mathbf{U}^{(\mathbf{1})}$, $\mathbf{U}^{(\mathbf{3})}$
2: **Output:** $\mathscr{P}_2 \in \mathbb{R}^{R_1 \times I_2 \times R_3}$
3: compute ${\mathbf{U}^{(\mathbf{1})}}^{-1}$ (use transpose for orthogonal matrices)
4: compute ${\mathbf{U}^{(\mathbf{3})}}^{-1}$ (use transpose for orthogonal matrices)
5: memory optimized version (after [Kiers, 2000]):
6: TTM (frontal FWD) multiplication of ${\mathbf{U}^{(\mathbf{1})}}^{-1}$ with $\mathscr{A} \rightarrow \mathscr{T}$ (Alg. 2, line 4)
7: TTM (lateral FWD) multiplication of ${\mathbf{U}^{(\mathbf{3})}}^{-1}$ with $\mathscr{T} \rightarrow \mathscr{P}_2$ (Alg. 2, line 5)

Algorithm 5 Optimize mode $n = 3$.

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, $\mathbf{U}^{(\mathbf{1})}$, $\mathbf{U}^{(\mathbf{2})}$
2: **Output:** $\mathscr{P}_3 \in \mathbb{R}^{R_1 \times R_2 \times I_3}$
3: compute ${\mathbf{U}^{(\mathbf{1})}}^{-1}$ (use transpose for orthogonal matrices)
4: compute ${\mathbf{U}^{(\mathbf{2})}}^{-1}$ (use transpose for orthogonal matrices)
5: memory optimized version (after [Kiers, 2000]):
6: TTM (frontal FWD) multiplication of ${\mathbf{U}^{(\mathbf{1})}}^{-1}$ with $\mathscr{A} \rightarrow \mathscr{T}$ (Alg. 2, line 4)
7: TTM (horizontal FWD) multiplication of ${\mathbf{U}^{(\mathbf{2})}}^{-1}$ with $\mathscr{T} \rightarrow \mathscr{P}_3$ (Alg. 2, line 5)

Algorithm 6 Derive core orthogonal factor matrices. TTM after [Kiers, 2000] (see Alg. 16, line 13).

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, $\mathbf{U^{(1)}}$, $\mathbf{U^{(2)}}$, $\mathbf{U^{(3)}}$
2: **Output:** $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$
3: set $\mathbf{U^{(1)}}^{-1} = \mathbf{U^{(1)}}^T$
4: set $\mathbf{U^{(2)}}^{-1} = \mathbf{U^{(2)}}^T$
5: set $\mathbf{U^{(3)}}^{-1} = \mathbf{U^{(3)}}^T$
6: TTM (frontal FWD) multiplication of $\mathbf{U^{(1)}}^T$ with $\mathscr{A} \rightarrow \mathscr{T}$
7: TTM (horizontal FWD) multiplication of $\mathbf{U^{(2)}}^T$ with $\mathscr{T} \rightarrow \mathscr{P}$
8: TTM (lateral FWD) multiplication of $\mathbf{U^{(3)}}^T$ with $\mathscr{P} \rightarrow \mathscr{B}$

Algorithm 7 Derive core non-orthogonal factor matrices (see Alg. 16, line 13).

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, $\mathbf{U^{(1)}}$, $\mathbf{U^{(2)}}$, $\mathbf{U^{(3)}}$
2: **Output:** $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$
3: compute $\mathbf{U^{(1)}}^+$ (use matrix pseudo inverse)
4: compute $\mathbf{U^{(2)}}^+$ (use matrix pseudo inverse)
5: compute $\mathbf{U^{(3)}}^+$ (use matrix pseudo inverse)
6: TTM (frontal FWD) multiplication of $\mathbf{U^{(1)}}^+$ with $\mathscr{A} \rightarrow \mathscr{T}$
7: TTM (horizontal FWD) multiplication of $\mathbf{U^{(2)}}^+$ with $\mathscr{T} \rightarrow \mathscr{P}$
8: TTM (lateral FWD) multiplication of $\mathbf{U^{(3)}}^+$ with $\mathscr{P} \rightarrow \mathscr{B}$

HOEIGS: Use EIG for HOSVD Implementation

As elaborated in App. B.2.1, the SVD for a symmetric input matrix equals its symmetric eigenvalue decomposition. Based on that the HOSVD can also be computed via symmetric eigenvalue decomposition what we call HOEIGS. In Fig. 7.5, the differences between the HOSVD and HOEIGS are visualized. The HOEIGS was implemented with a *vmmlib* wrapper, which uses the LAPACK DSYEVX (symmetric eigenvalue decomposition) to return the first R largest magnitude eigenvalues with corresponding eigenvectors. These eigenvectors correspond in the $\mathbf{C}_{(n)} = \mathbf{A}_{(n)}\mathbf{A}_{(n)}^T$ covariance matrix scenario to the R first left singular vectors that are used as TA factor matrices $\mathbf{U^{(n)}}$ (see Alg. 8). For the covariance matrix computation, again BLAS DGEMM was used.

To further save computing time and memory, we initialize the factor matrices for the ALS with random values as in [Wang et al., 2005b]. Neither [Wang et al., 2005b] nor we noticed an empirical difference when initializing the factor matrices by random values. On the contrary, the simple random factor matrix initialization in fact improves the computation time. In particular, the first factor matrix computation – usually generated by the HOSVD itself – is the most expensive step. This is because this initial HOSVD would be computed on a large data tensor $\mathscr{A}_{I_1 \times I_2 \times I_3}$, where subsequent HOSVDs during the ALS iterations only work on tensors $\mathscr{T}_{I \times I \times R}$ or $\mathscr{P}_{I \times R \times R}$ with $R \ll I$.

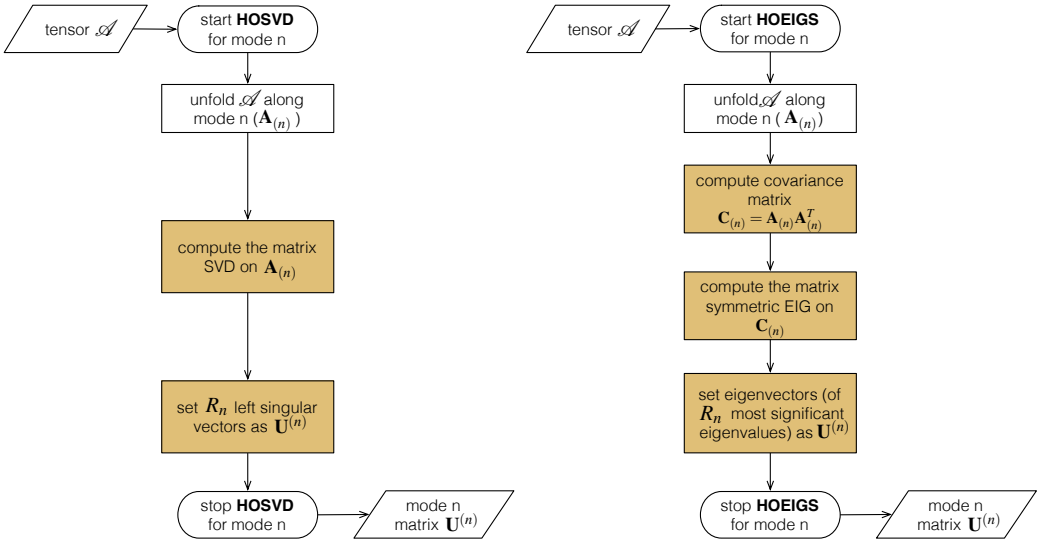


Figure 7.5: *Visualization of the HOSVD and the HOEIGS implementations.*

Algorithm 8 HOSVD by EIGS along mode n.

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, R_n
2: **Output:** optimized $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$
3: **for** mode n of N **do**
4: unfold $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into its matrix representation $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdot \ldots \cdot I_{n-1} \cdot I_{n+1} \cdot \ldots \cdot I_N)}$
5: build the covariance matrix $\mathbf{C}_{(n)} \in \mathbb{R}^{I_n \times I_n}$ from $\mathbf{A}_{(n)}\mathbf{A}_{(n)}^T$
6: compute the EIG $\mathbf{C}_{(n)} = \mathbf{Q}^{(n)}\Lambda\mathbf{Q}^{(n)^T}$
7: sort the eigenvalues λ_j and its corresponding eigenvectors $\boldsymbol{q}_j^{(n)}$ by largest magnitude
8: set the R_n sorted eigenvectors $\mathbf{Q}^{(n)'}$, which correspond to the leading left singular values, to the mode-n factor matrix $\mathbf{Q}^{(n)'} \rightarrow \mathbf{U}^{(n)}$
9: **end for**

The choice of the initial rank R_{init} is an important factor of the computational cost of the initial tensor decomposition. The smaller R_{init}, the fewer computations in bottleneck (1) and bottleneck (2) are needed. The typical setting for a reduced rank TA would be half of the dimension of the input volume $\mathscr{A}$, thus $R_{init} = \frac{I}{2}$ (see [Wu et al., 2008; Suter et al., 2010a; Suter et al., 2011]). However, in one of the developed TA multiresolution models, [Suter et al., 2013], it was possible to reduce R_{init} to half of the brick size instead. This is a great benefit once the initial datasets get large.

Achieved Decomposition Times

For performance analysis, the critical steps during the initial tensor decomposition steps were measured. The tensor decomposition has been carried out on an Quad-Core Intel Xeon 2.4GHz MacPro5,1 with 22GB RAM and a 500GB SSD hard disk, and a Geforce GTX 285 graphics card with 1GB memory. In Tab. 7.2, the total decomposition time for HOSVD and HOEIGS as well as the time for the most critical TTM (Fig. 3.14(a)), the SVD, the EIGS, and the covariance matrix are shown. Different data sizes were used to give insight into the scaling of the initial decomposition algorithm steps. As a conclusion from Tab. 7.2, it can be seen that the SVD/EIGS computation and the critical TTM computation are the most expensive steps. The critical TTM is used once per ALS iteration, while the SVD/EIGS is used three times per ALS iteration. Regarding the data size of the input tensor, it can be derived that the HOSVD is in advantage for large initial tensor decomposition, while for data bricks or small volumes, the HOEIGS is marginally faster. Whereas [Kolda et al., 2008] reported that for MATLAB tensor toolbox and for a tensor $\mathscr{A}_{100^3}$ the HOEIGS was roughly 10 times faster than the HOSVD. However, it has to be considered that the computation is highly dependent on the used HOEIGS, HOSVD, LAPACK and BLAS implementation as well as the computer hardware.

After the tensor decomposition representation implementation has been introduced, the generation of the multiresolution data structure is illustrated.

7.3 Octree Build

As introduced in Chap. 4, a multiresolution model is an essential part for direct volume rendering of large datasets. As underlying multiresolution data structure an octree with equally sized nodes (bricks or subvolumes) was chosen. This octree data structure is produced in an “offline” preprocessing routine prior to rendering. During this thesis a generic octree build was implemented for the multiresolution model with global mipmapped TA bases (see Fig. 4.1(b)). The described generic

	Brick $\mathscr{A}_{64^3}$	Bonsai $\mathscr{A}_{256^3}$	Hnut $\mathscr{A}_{512^3}$	Flower $\mathscr{A}_{1024^3}$	Branch 1 $\mathscr{A}_{2048^3}$	Branch 2 $\mathscr{A}_{2048^2\times4096}$
HOSVD	$0.13s$	$1.6s$	$8.8s$	$50s$	$15min$	20min
HOEIGS	$0.11s$	$1.6s$	$10s$	$70s$	$28min$	$83min$
Critical TTM	$0.003s$	$0.03s$	$0.3s$	$2.5s$	$2.4min$	$4.7min$
Other TTMs	$\leq 0.001s$	$\leq 0.01s$	$\leq 0.05s$	$\leq 0.2s$	$\leq 7s$	$\leq 15s$
SVD	$\leq 0.01s$	$\leq 0.13s$	$\leq 0.65s$	$\leq 3.5s$	$\leq 8s$	$\leq 4.5min$
EIGS	$\leq 0.005s$	$\leq 0.10s$	$\leq 0.70s$	$\leq 5s$	$\leq 42s$	$\leq 5.5min$
$\mathbf{C} = \mathbf{A}\mathbf{A}^T$	$0.002s$	$0.01s$	$0.05s$	$0.2s$	$0.8s$	$3.5s$

Table 7.2: *Times of different steps during the initial tensor decomposition, measured with differently sized datasets. All initial decompositions were run over three ALS iterations.*

octree build was used in [Suter et al., 2013] and is explained in more detail here. Prior to the actual octree node generation, the global mipmapped TA bases need to be generated.

7.3.1 Processing of Global TA Bases

In this section, the algorithm (Alg. 9) to produce the global mipmapped factor matrices is given. We start with the initial three factor matrices that were produced on the full input dataset (see Alg. 9, line 3) and average them to all lower resolution octree hierarchy levels (see Fig. 7.6 and Alg. 9, line 5). As described in Sec. 6.3, the submatrices in our mipmapped global TA matrices need to fulfill certain orthogonality constraints in order to produce rank-reducible core tensors. Therefore, as shown in Fig. 7.7 and Alg. 9, line 9, another SVD on a row-subset of the initial matrices is applied.

Algorithm 9 Generate global TA bases.

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$
2: **Output:** $\mathbf{U}^{(n)'}_{J_n\downarrow_k} \in \mathbb{R}^{(\frac{I_n}{k+1} + x \cdot 2 \cdot W_n - 2 \cdot W_n) \times R_n}$
3: generate initial factor matrices $\mathbf{U}^{(\mathbf{n})}$ by the HOOI (Alg. 16)
4: **for** every octree level k **do**
5: downsample factor matrices to $\mathbf{U}^{(\mathbf{n})}_{\downarrow_\mathbf{k}}$
6: recompose factor matrices
7: **for** every row-block s_n of brick length $J_n = B + W_n$ **do**
8: get submatrix $\mathbf{U}^{(n)}_{J_n\downarrow_k}$ (including brick borders)
9: get new $\mathbf{U}^{(n)'}_{J_n\downarrow_k}$ via SVD on $\mathbf{U}^{(n)}_{J_n\downarrow_k}$
10: save processed matrices $\mathbf{U}^{(n)'}_{J_n\downarrow_k}$
11: **end for**
12: **end for**

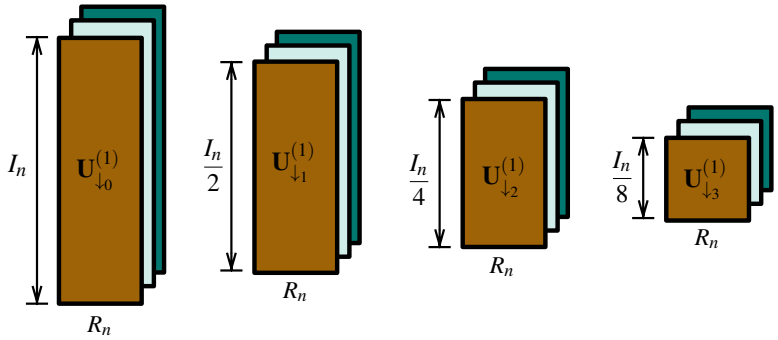


Figure 7.6: *Step one during factor matrix processing (spatial subsampling): Mipmapping of global TA factor matrices (see Alg. 9, line 5).*

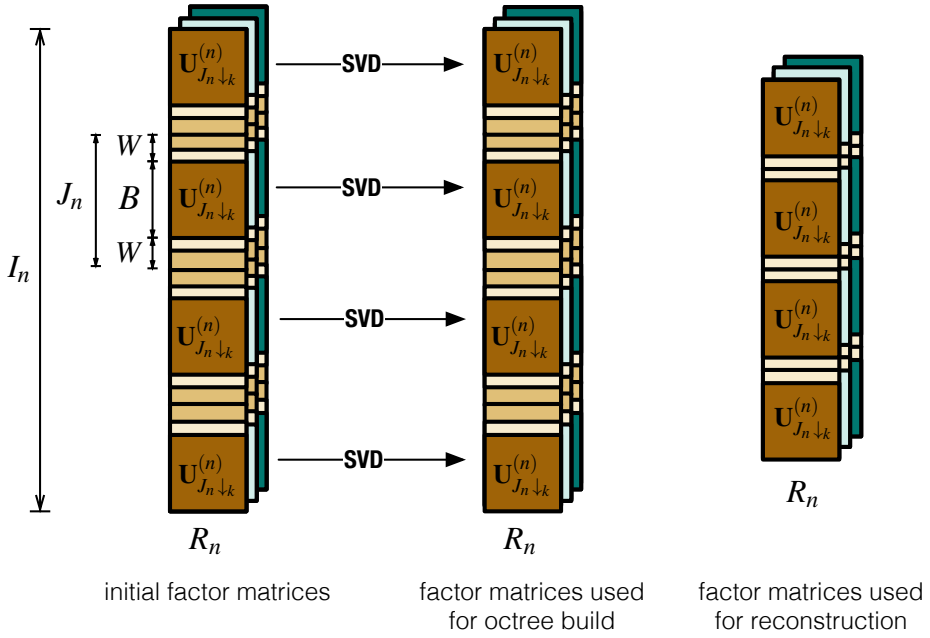


Figure 7.7: *Step two during factor matrix processing (spatial selection): Row-block SVDs corresponding to bricks of the global TA factor matrices (see Alg. 9, lines 7–11).*

Fig. 7.8 visualizes the submatrices $\mathbf{U}^{(n)}_{J_n\downarrow_k}$ for a given brick of size $B = 64$ and a given octree level k, as they need to be selected in Alg. 9, line 8. During the row-subset SVD, the row-blocks are chosen for each brick and its available borders W_n. As described in Sec. 6.3 we use a 2-voxel border for the volume rendering (gradient interpolation) and a 4-voxel border for the information overlap between neighboring bricks (see also Fig. 7.7). For the submatrix with length $J_n = B + W_n$ corresponding to a brick with index i_n that is:

$$W_n = \begin{cases} W_n = 0 & \text{if } ((i_n == 0) \text{ AND } (i_n == (\frac{I_n}{B} - 1)) \\ W_n = 2+4 = 6 & \text{if } ((i_n == 0) \text{ AND } (i_n < (\frac{I_n}{B} - 1)) \\ W_n = 2+4 = 6 & \text{if } ((i_n > 0) \text{ AND } (i_n == (\frac{I_n}{B} - 1)) \\ W_n = 2 \cdot (2+4) = 12 & \text{if } ((i_n > 0) \text{ AND } (i_n < (\frac{I_n}{B} - 1)). \end{cases}$$

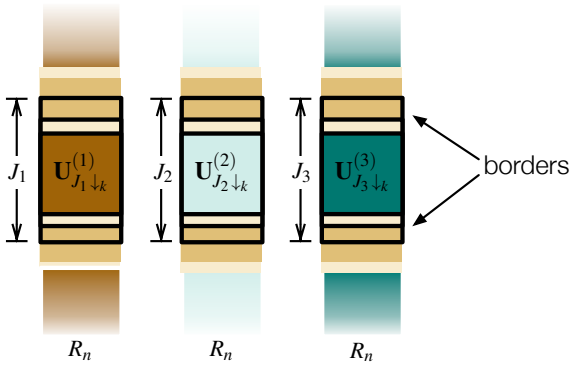


Figure 7.8: *Submatrices $\mathbf{U}^{(n)}_{J_n\downarrow_k}$ for a given brick of size $B = 64$ and a given octree level k: the submatrix length J_n includes brick borders needed for the core tensor generation. The borders W_n are visualized with the light brown colors.*

To give an idea what such global TA bases look like, we visualize in Fig. 7.9 the global factor matrices of $\mathbf{U}^{(1)}$ of the hazelnut dataset. The intensity distributions look similar to frequency patterns but in fact show the input-data-specific distribution of the TA's data-specific factor matrix bases. Furthermore, similar to a matrix PCA the first rank is represented by one major base frequency, while the frequencies increase with subsequent ranks, i.e., higher frequency details are encoded with additional ranks.

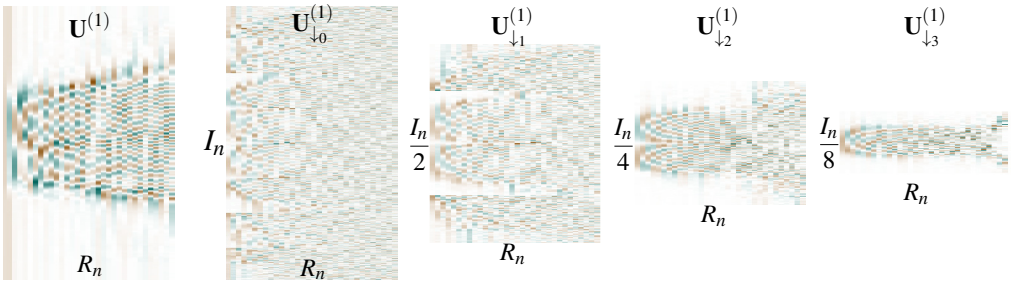


Figure 7.9: *Visualization of an initial factor matrix $\mathbf{U}^{(1)}$ of the hazelnut and its full resolution row-block SVD replacement $\mathbf{U}^{(1)}_{\downarrow_0}$. Subsampled matrices $\mathbf{U}^{(1)}_{\downarrow_k}$ are stretched to fit and value coded: brown (negative), white (zero), green (positive), [Suter et al., 2013].*

Based on these global mipmapped TA factor matrices, each brick can be compressed with submatrices defined in this section and as described next.

7.3.2 Brick Compressor

In order to make a link between the global TA factor matrices and the actual multiresolution data, the octree data structure is generated. As shown in Fig. 7.10, an octree node in our TA multiresolution model consists of one core tensor. Alg. 10 and Fig. 7.11 describe the procedure to produce one core tensor or one octree node (the so-called brick compression). The basic steps of the brick compression are: (1) load a brick $\mathscr{A}_{brick}$ at position (i_1, i_2, i_3) with available borders, (2) load the corresponding submatrices, (3) compute the core tensor (projection of brick onto inverse submatrices), and (4) encode the core tensor with an 8-bit logarithmic quantization. The generic octree build is the topic of the next section.

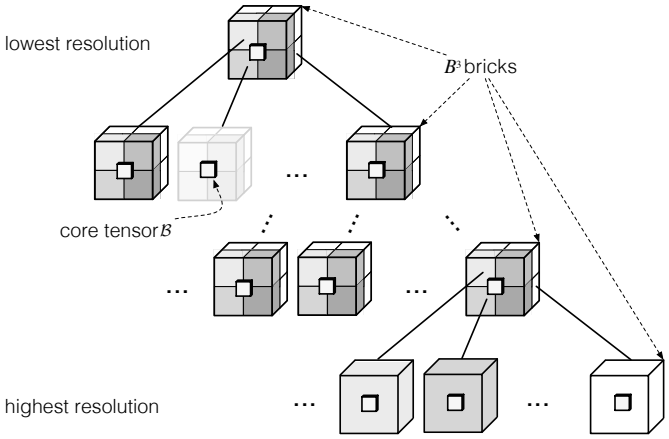


Figure 7.10: *TA-based octree data structure: One octree node is represented with a core tensor showing the relationship to the global TA factor matrices and the original data.*

Algorithm 10 Brick compressor: Core tensor generation from global TA matrices for a given hierarchy level k and a brick $\mathscr{A}_{\text{brick}\downarrow k}$ at position (i_1, i_2, i_3).

1: **Input:** brick data and borders $\mathscr{A}_{\text{brick}\downarrow k} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ at position (i_1, i_2, i_3)
2: **Output:** quantized core tensor $\mathscr{B}_{\text{brick}\downarrow k} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ for position (i_1, i_2, i_3)
3: **if** $\mathscr{A}_{\text{brick}\downarrow k}$ is empty **then**
4: skip this brick
5: **else**
6: determine submatrices $\mathbf{U}^{(n)}_{J_n \downarrow_k}$ for position (i_1, i_2, i_3)
7: compute core tensor:
$\mathscr{B}_{\text{brick}\downarrow k} = \mathscr{A}_{\text{brick}\downarrow k} \times_1 \mathbf{U}^{(1)\ T}_{J_n \downarrow_k} \times_2 \mathbf{U}^{(2)\ T}_{J_n \downarrow_k} \times_3 \mathbf{U}^{(3)\ T}_{J_n \downarrow_k}$
8: quantize core (log2, float32 to uint8) and store it with an id
9: **end if**

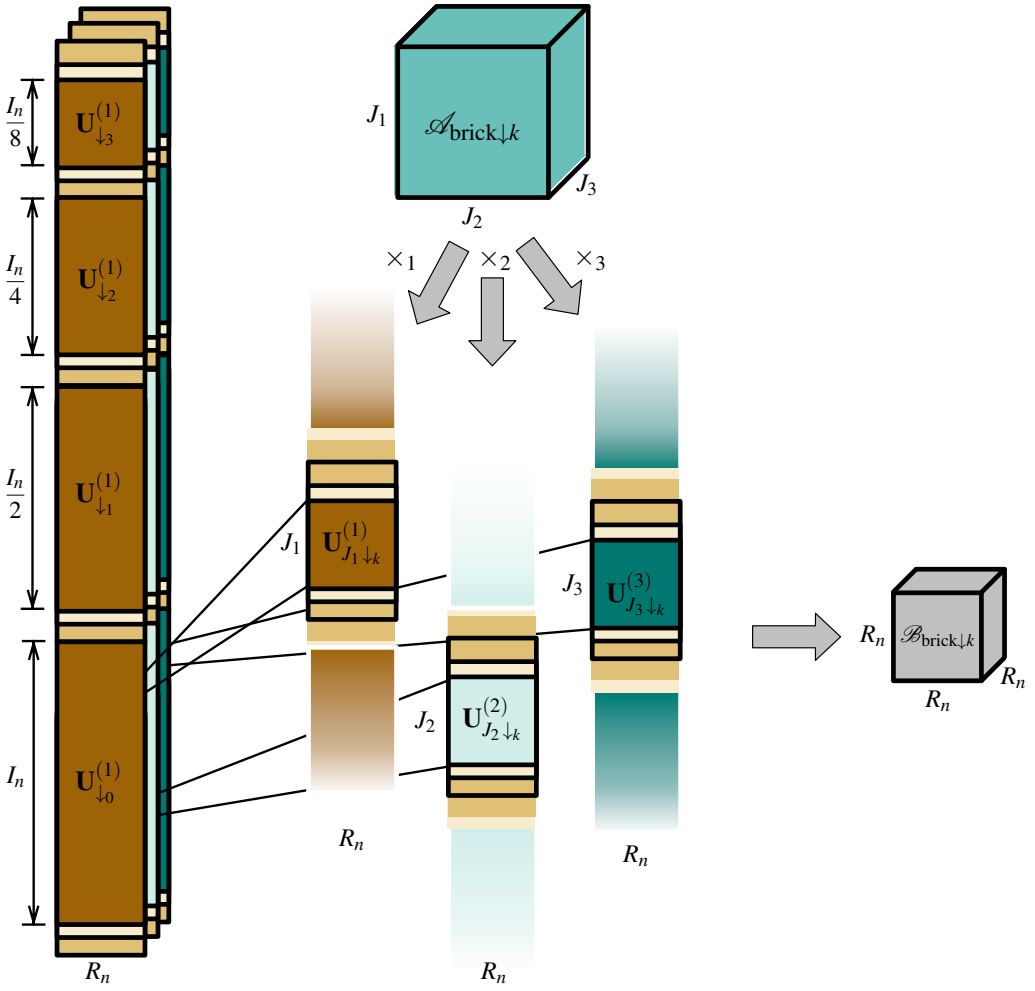


Figure 7.11: *Load submatrices from global TA factor matrices in order to project the brick on the submatrices. The projection forms the core tensor of one octree node.*

7.3.3 Generic Octree Building

The generic octree data structure building for the described multiresolution model is shown here. The data of an octree node corresponding to an input subvolume $\mathscr{A}_{J_1 \times J_2 \times J_3}$ is given by its core tensor $\mathscr{B}_{R_1 \times R_2 \times R_3}$, whose coefficients indicate the relationship between the factor matrices and the volume.

Given the mipmapped factor matrices $\mathbf{U}^{(n)}_{\downarrow_k}$, with R columns each, we compute the core tensor $\mathscr{B}_{R \times R \times R}$ per octree node. That is, the input subvolume brick $\mathscr{A}_{J_1 \times J_2 \times J_3}$, at the required spatial resolution, is projected onto the row-block SVD factor submatrices $\mathbf{U}^{(n)}_{J_n \downarrow_k}$ of the appropriate subsampling level k(see Sec. 6.3). The length J_n of each brick $\mathscr{A}_{brick}$ includes available brick borders W_n as defined in Sec. 7.3.1. Without restricting the generality of the described concepts, in the following we assume a subvolume brick size of $B^3 = 64^3$, thus per-brick core tensor ranks $R = 32$ and core tensors of size R^3. The core tensor rank R is equal to the initially chosen rank R_{init} of the global mipmapped TA factor matrices, i.e., $R = R_{init} = 32$. For empty bricks, the node's core tensor is null and skipped.

The implementation of the generic octree building is performed block-wise, as illustrated in Fig. 7.12. One block is of size $\mathscr{A}_{I_1 \times I_2 \times J_3}$, where J_3 is the size along the third spatial data axis. These blocks are loaded and processed one after the other. For every data block the core tensors of all bricks are generated, encoded and stored. Moreover, the data blocks are averaged to all their lower resolution octree hierarchy levels that are maintained in a separate array in the main memory. Once the lower level data bricks are filled, they are processed, too. This temporary data array stores for every octree hierarchy level averaged data blocks corresponding to four brick sizes ($I_{1\downarrow_k} \times I_{2\downarrow_k} \times 4 \cdot B$). By maintaining four averaged bricks per octree level the access to neighboring bricks for the border collection is guaranteed. Once the averaged bricks are processed, the two last averaged blocks at position 2 and 3 are moved to the start of this octree level and the next loaded block is downsampled to position 2. This procedure is continued until all data blocks are loaded and all octree levels are processed. The octree building process is shown in Alg. 11.

Note: The node ids for the octree data structure are given level-wise. The root level has always the node id 1, the second level has always the node ids 2 to 9. The starting node of a level k can be computed as $id = \sum_{i=0}^{k-1} 2^{3 \cdot i}$. For each octree level, the ids are given per block. The ids are first enumerated along the $I_{1\downarrow_k}$ axis and then along the $I_{2\downarrow_k}$ axis, as visualized in Fig. 7.13. For the visualization system, a data structure accessing the bricks z-curve indexing is constructed such that each child node can be access via $child\_id = (parent\_id \cdot 8) + 1$ and vice versa via $parent\_pos = \frac{child\_pos - 1}{8}$. A converter creates the desired node id transformation for the visualization system and stores the individual cores in 2GB files.

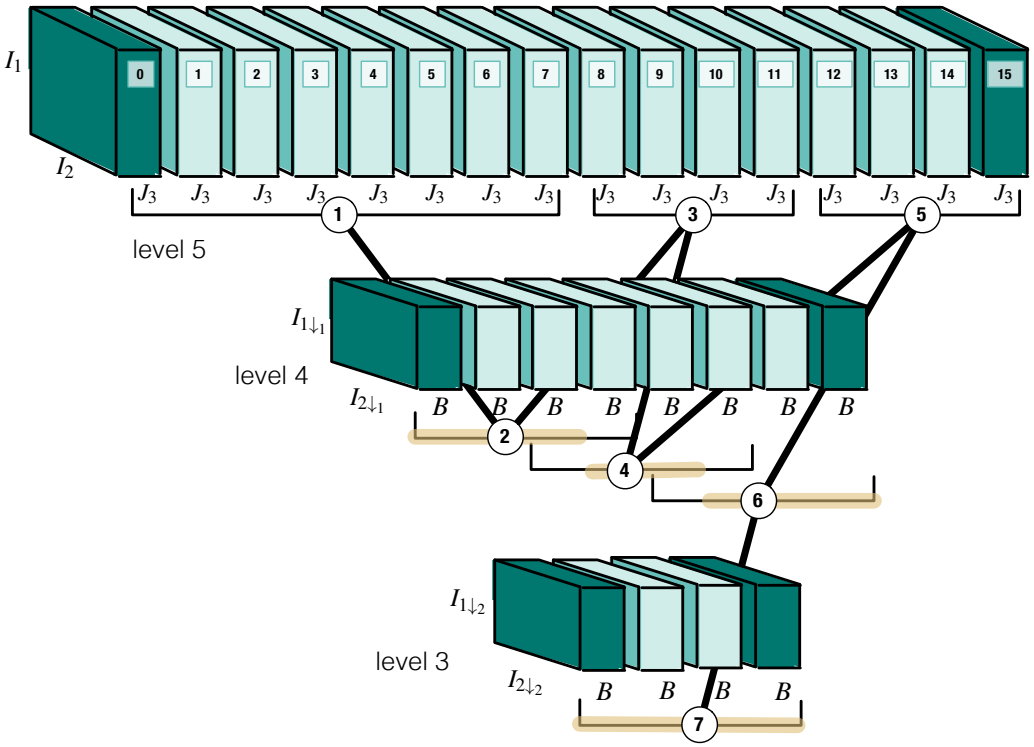


Figure 7.12: *Octree averaging on the fly: Illustrated on a five level octree hierarchy with a brick size of $B = 64$. The original data is loaded as blocks of size $I_1 \times I_2 \times J_3$, where J_3 is the brick size B plus available borders (top row). Whenever new data blocks are loaded, they are downsampled to their lower resolution representations (middle and bottom row). For the lower resolution representations, always four blocks of size $I_{1\downarrow_k} \times I_{2\downarrow_k} \times 4 \cdot B$ are kept in memory. In order to have brick borders available at the downsampled blocks, the brick borders need to be collected from the neighboring blocks. Once the first eight blocks are loaded, the data in all the lower resolution representation is shifted leftwards by two blocks before the next four full resolution blocks are loaded. This procedure guarantees to have all the necessary brick borders available for the lower resolution representations. Thus, in the lower resolution representations, the number of blocks that are processed depends on their position and on their availability of brick borders. The circled numbers show in what order the data blocks are processed in order to produce the core tensor hierarchy.*

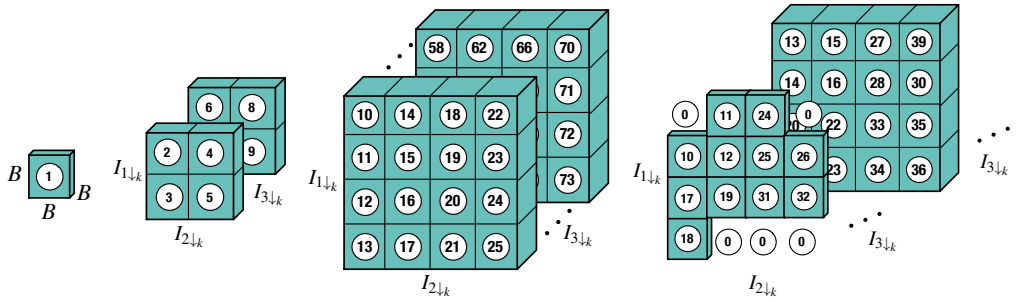
Algorithm 11 Generic octree build.

Input: $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$
Output: multiresolution TA data structure (matrices; core tensors) and meta information file
process factor matrices, as shown in Alg. 9
for every data block **do**
    load data block $\mathscr{A}_{I_1 \times I_2 \times J_3}$ into memory
    average data block to lower resolutions
    for all brick $\mathscr{A}_{\text{brick}\downarrow k}$ in leaves' level of the octree hierarchy, i.e., k = 0 **do**
        compress brick as in Alg. 10
    end for
    if the first eight full resolution blocks were loaded **then**
        for every brick $\mathscr{A}_{\text{brick}\downarrow k}$ in the first three downsampled blocks **do**
            compress brick (Alg. 10)
        end for
    end if
    if last blocks were loaded **then**
        for every brick $\mathscr{A}_{\text{brick}\downarrow k}$ in the last three downsampled blocks (if available) **do**
            compress brick (Alg. 10)
        end for
    end if
    if inner blocks were loaded **then**
        for every brick $\mathscr{A}_{\text{brick}\downarrow k}$ in the two middle downsampled blocks **do**
            compress brick (Alg. 10)
        end for
    end if
end for
for every brick $\mathscr{A}_{\text{brick}\downarrow k}$ in the level above the root **do**
    compress brick as in Alg. 10
    average data to root brick
end for
for root brick $\mathscr{A}_{\text{brick}\downarrow k}$ **do**
    compress root brick as in Alg. 10
end for
save meta information file

Finally, for the multiresolution DVR and multiscale feature visualization coupling as described in Sec. 6.3, an LOD error has to be computed per brick. This procedure is performed after the octree is built and is described next.

LOD Error Pre-computation

As part of the preprocessing – after the octree build, a per-brick error is computed in order to couple the rank-reduced reconstructions with an error corresponding to different feature scale renderings (see Sec. 6.1.1). The RMSEs for a certain number of rank-reduced brick reconstructions are stored in a separate file. Each brick RMSE is computed using trilinear interpolation to the original LOD. During the



(a) octree build (b) visualization system octree.

Figure 7.13: *Octree ids illustrated for three octree levels.*

interactive visualization, the user can decide what error, i.e., what approximation level, he allows.

After having introduced the generation of the multiresolution data structure (octree), in the next section, the reconstruction strategy from the multiresolution DVR structure is presented. While the octree building is not time-critical, it is a need that the reconstruction process performs at run-time.

7.4 Parallelization of the Tensor Reconstruction

7.4.1 GPGPU-based Tensor Reconstruction

For volume ray casting one can consider either a per-voxel (e.g., for random access during traversal) or per-brick reconstruction approach. Using a per-brick solution permits us to optimize reconstruction by refactoring computations in order to reduce computational costs and to take advantage of the complex memory hierarchy of nowadays GPGPU platforms. The reconstruction is performed brick-wise. Each requested brick, if not already cached by the rendering system, is loaded and transferred to the GPU where the tensor reconstruction is performed. The GPGPU-based tensor reconstruction strategy and implementation was published in [Suter et al., 2011]. The basic concepts of the reconstruction were, however, applied to both of the multiresolution models from Chap. 4. In the following, the CUDA implementation for the optimized Tucker reconstruction by TTMs (Sec. 3.6) is explained. Before that, the CUDA terminology is briefly introduced.

7.4.2 CUDA Terminology

A CUDA kernel is a SIMD parallel program that is executed by an array of *threads* (work-items),[1] all running the same code. The threads are organized in a grid of thread blocks, and each kernel is called with a given *grid* size (NDRange) and a given *blocks* size (work groups). Each thread owns some *registers* (private memory) and each thread block has access to a limited amount of *shared memory* (local memory), which constitute the fastest data access paths. Additionally all threads can concurrently access *global memory*, *constant memory* and *texture memory* (global memory), where the memory latency increases from constant and texture memory (read only and cached), to local and global memory (read/write).

7.4.3 CUDA TTM Reconstruction

The reconstruction process from our chosen tensor decomposition is, in principle, straightforward, and can be optimized by a careful reordering of operations (Sec. 3.6). Nevertheless, reconstruction time can be critical for real-time visualization and therefore needs to be given attention in our system. For the first time, we address GPGPU-based tensor reconstruction. The reconstruction is expected to be faster when following a strategy that sticks to general concepts of parallel computing. For example, while thresholding of tensor coefficients [Wu et al., 2008] may reduce the amount of data, the reconstruction process can be more tedious for parallel computing since complex decoding algorithms have to be used. Moreover, the reconstruction is affected by the format and representation of the coefficients.

The tensor reconstruction in CUDA is implemented using three successively applied TTM kernels. Each kernel corresponds to the application of one n-mode product (Eq. (C.3)), hence TTM1, TTM2, and TTM3. After applying TTM1 and then TTM2, we need each time to temporarily store a 3^{rd}-order tensor of size $B \times R^2$ and $B^2 \times R$, respectively. Eventually after applying TTM3 the final B^3 sized volume brick is reconstructed. The pseudo code implementation of the three CUDA kernels is given in the Algs. 12–14. We use a thread block per decoded brick, where each thread is responsible for computing one element of the tensor-matrix multiplication. The grid-size for parallel execution is determined by the number of bricks that need to be decoded in the current frame (e.g., 8 bricks for the minimal octree refinement).

The simplified CUDA implementation layout is shown in Fig. 7.14. For TTM1, we compute one slice of the intermediate tensor $\mathscr{B}'$ on one thread block. For one TTM1-block, we load the factor matrix $\mathbf{U}^{(1)}$ and one slice of $\mathscr{B}$ (slice is given by blockId). For reasons of memory optimizations, we compute for TTM2 and

[1] Analogous OpenCL terms are mentioned in parenthesis.

TTM3 only half slice of $\mathscr{B}''$ and half slice of $\widetilde{\mathscr{A}}$, respectively, on one thread block. For one TTM2/TTM3-block, we load half of the factor matrix and one slice of the intermediate data structures $\mathscr{B}'$ and $\mathscr{B}''$. The memory usage and performance optimizations of the CUDA TTM reconstruction, following CUDA implementation best practices [CUD, 2010], are explained next.

Algorithm 12 CUDA kernel for TTM1. From [Suter et al., 2011].

load $\mathbf{U}^{(\mathbf{1})}$ and tensor core $\mathscr{B}$ slices to GPU
CUDA kernel:
extract min/max values for dequantization
linearly dequantize one element of $\mathbf{U}^{(\mathbf{1})}$
log dequantize one element of $\mathscr{B}$
each thread writes one element of $\mathbf{U}^{(\mathbf{1})}$ to shared memory
each thread writes one element of the $\mathscr{B}\_slice$ to shared memory
__synchthreads()
for each r1 in R1 **do**
    voxel += $\mathbf{U}^{(\mathbf{1})}(i1, r1) \cdot \mathscr{B}(r1, r2, r3)$
end for
store *voxel* to the intermediate $\mathscr{B}'(i1, r2, r3)$

Algorithm 13 CUDA kernel for TTM2. From [Suter et al., 2011].

load $\mathbf{U}^{(\mathbf{2})}$ and half tensor $\mathscr{B}'$ slices to GPU
CUDA kernel:
extract min/max values for dequantization
linearly dequantize one element of $\mathbf{U}^{(\mathbf{2})}$
each thread writes one element of $\mathbf{U}^{(\mathbf{2})}$ to shared memory
each thread writes one element of the $\mathscr{B}'\_slice$ to shared memory
__synchthreads()
for each r2 in R2 **do**
    voxel += $\mathbf{U}^{(\mathbf{2})}(i2, r2) \cdot \mathscr{B}'(i1, r2, r3)$
end for
store *voxel* to the intermediate $\mathscr{B}''(i1, i2, r3)$

Algorithm 14 CUDA kernel for TTM3. From [Suter et al., 2011].

load $\mathbf{U}^{(\mathbf{3})}$ and half tensor $\mathscr{B}''$ slices to GPU
CUDA kernel:
extract min/max values for dequantization
linearly dequantize one element of $\mathbf{U}^{(\mathbf{3})}$
each thread writes one element of $\mathbf{U}^{(\mathbf{3})}$ to shared memory
each thread writes one element of the $\mathscr{B}''\_slice$ to shared memory
__synchthreads()
for each r3 in R3 **do**
    voxel += $\mathbf{U}^{(\mathbf{3})}(i3, r3) \cdot \mathscr{B}''(i1, i2, r3)$
end for
add contribution of hot-corner core value to *voxel*
store *voxel* in decoding buffer for $\widetilde{\mathscr{A}}(i1, i2, i3)$

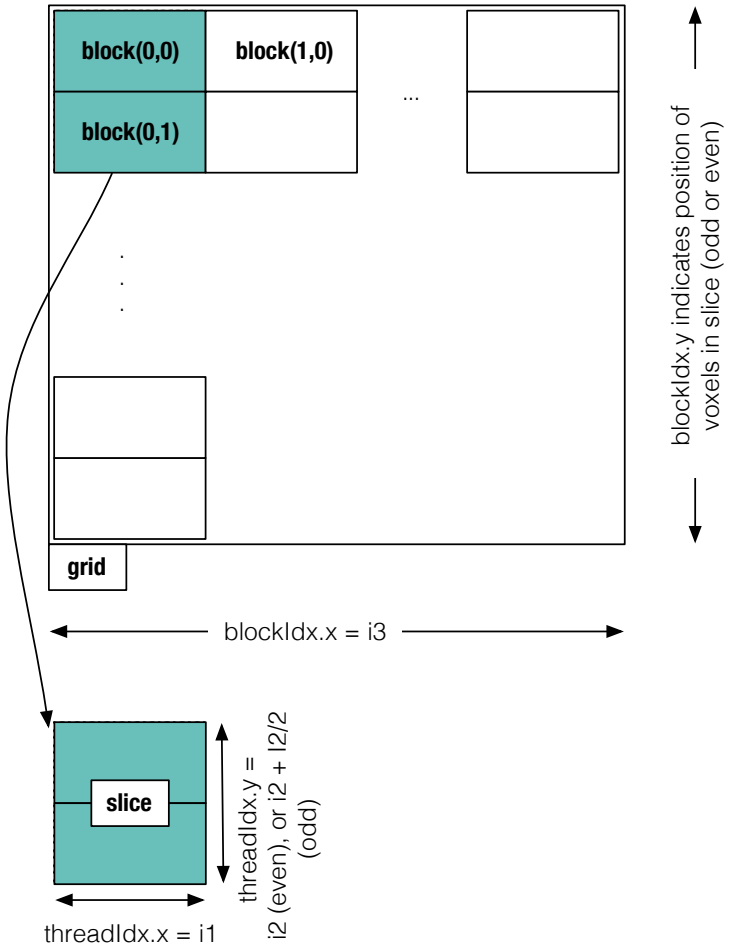


Figure 7.14: *CUDA implementation layout of tensor reconstruction: One CUDA thread computes one voxel $a_{i_1 i_2 i_3}$. Otherwise a slice-wise matrix matrix computation was chosen such that it fits the grid size and layout of CUDA.*

7.4.4 Performance Optimizations

In order to optimize the parallel execution on the GPU, data parallelism and memory usage should be optimized, taking into account the bandwidths to the parts of the memory hierarchy, thus increasing throughput. In our approach, host-to-device data transfers are reduced by using page-locked buffers. That is, before launching the kernel TTM1, the quantized factor matrices and the quantized core tensor of one brick are transferred all at once to the global GPU memory. Our GPU TTM reconstruction code (TTM kernels) uses intermediate data structures ($\mathscr{B}'$ and $\mathscr{B}''$), which allows us to make use of the on-chip memory.

The temporary data structures of $\mathscr{B}'$ and $\mathscr{B}''$ are stored in the global memory, and slices of these 3^{rd}-order tensors are loaded to the shared memory when requested. Threads within the same thread block cooperate to load into shared memory the necessary elements of $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$,$\mathbf{U}^{(3)}$, $\mathscr{B}$, $\mathscr{B}'$ and $\mathscr{B}''$. Always, one single tensor element is loaded per thread. A *syncthreads()* barrier at the end of the loading phase ensures that all the elements are up-to-date before performing calculations. In order to avoid bank conflicts in shared memory accesses, the factor matrices and core tensors were dequantized to 32-bit floating point words before we upload the data to shared memory. Thus, all accesses are aligned on 32-bit words.

For TTM2 and TTM3, we split the matrix matrix multiplication of one slice into two blocks. In that way, we optimize the shared memory usage and load only half of a factor matrix together with the full core tensor (we thus compute only upper or lower half of a matrix with the other matrix). With this scenario, we need for TTM1 $(B \cdot R \cdot 4 + R \cdot R \cdot 4)$ bytes and for TTM2/TTM3 $(B/2 \cdot R \cdot 4 + B \cdot R \cdot 4)$ bytes of shared memory per block, which works with 32-cubed bricks for CUDA 1.x and 2.x. We have a maximum of 16 KB (CUDA 1.x) or 48 KB (CUDA 2.x) of shared memory available per multiprocessor, where one multiprocessor can have at maximum 8 blocks. Depending on how much shared memory is used, fewer or more blocks are loaded per multiprocessor. To summarize, in our implementation, increasing the use of shared memory has higher priority than maximizing the number of blocks run per processor (for CUDA 1.x).

Based on this tensor-specific GPU-based reconstruction implementation, the visualization for the interactive rendering was performed. In the next section, the visualization system and the interactive performance for the tensor-based visualization are given.

7.5 Visualization and Interactive Performance

The two multiresolution models from Chap. 4 were implemented with two different systems and two different viewers. In the first model, the focus lies on

the interactivity of the advanced TTM reconstruction described in Sec. 7.4, in the second model, the focus lies on the coupling of the multiscalability and the multiresolution by the LOD error, as indicated in Sec. 6.3. Next, the two applications and their interactive performance are elaborated.

7.5.1 Application 1: TTM Reconstruction Performance

The bricked TA octree hierarchy as published in [Suter et al., 2011] was used to verify the GPU-based TTM reconstruction performance. First, the visualization system is described, then, the test setup to measure the interactive performance is outlined, and finally, the performance analysis is given.

At run-time, an adaptive loader updates a view-dependent and transfer function-dependent working set of bricks. The working set is incrementally maintained on the CPU and GPU memory by asynchronously fetching data from the out-of-core brick multiresolution TA structure. Following the MOVR approach [Gobbetti et al., 2008; Iglesias Guitián et al., 2010], the working set is maintained by an adaptive refinement method guided by the visibility information fed back from the renderer. The adaptive loader maintains on GPU a cache of recently used volume bricks, stored in a 3D texture. At each frame, the loader constructs a spatial index for the current working set in the form of an octree with neighbor pointers.

For rendering and visibility computation, the octree is traversed using a CUDA stackless octree ray caster, which employs preintegrated scalar transfer functions to associate optical properties to scalar values, and supports a variety of shading modes [Iglesias Guitián et al., 2010]. The ray caster works on reconstructed bricks, and reconstruction steps occur only upon GPU cache misses. The quantized tensor decomposition is dequantized and reconstructed on demand by the adaptive loader during the visualization on the GPU (see Sec. 7.4).

In order to make structural exploration of the datasets possible, the reconstruction can consider only the K most significant ranks of the tensor decomposition, where $K \in [1..R]$ is chosen by the user. The reconstruction rank K can be changed by the user during the visualization process with a *rank slider*. Lower-rank approximations give an outline of the visualized dataset and can highlight structures at specific scales [Suter et al., 2010a] (see Sec. 6.2). Higher K values add more details onto the dataset.

We implemented a library supporting the volume tensor reconstruction and its integration with a GPU-accelerated out-of-core multiresolution volume rendering framework using C++ and CUDA 3.2. All performance tests have been carried out on an Intel Core 2 E8500 3.2GHz Linux PC with 4GB RAM, and GeForce GTX 480 graphics with 1.5GB of memory. The multiresolution volume octree is stored in an out-of-core structure, based on the Berkeley DB, with each 32^3 brick being stored as a quantized rank-$(16,16,16)$ tensor decomposition. Preprocess-

ing consisted in the construction and storage of the multiresolution volume octree, including the computation of the tensor decomposition for all bricks and the quantization of the coefficients. The preprocessing time for the 2GB chameleon dataset was $25min15sec$ and in the case of the 16GB tooth dataset the preprocessing time was $8h45min$.

We evaluated the rendering and tensor reconstruction performance on the great ape molar (App. A). Our interactive inspection sequences include overall views and extreme close-ups, which stress our adaptive loader by incrementally requesting and reconstructing a large number of bricks. Fig. 7.15 demonstrates the achieved performance. As we can see, in any case an interactive rendering performance can be maintained, with frame-rates higher than 12Hz even for the most demanding situations, and on average between 50Hz and 100Hz. In particular, the timing reveals that our tensor reconstruction constitutes only a negligible overhead with respect to the overall rendering cost. Rendering time is in fact dominated by the ray casting and data transfer times. The most costly part of the tensor reconstruction process is the final copy of the decoded bricks to texture cache.

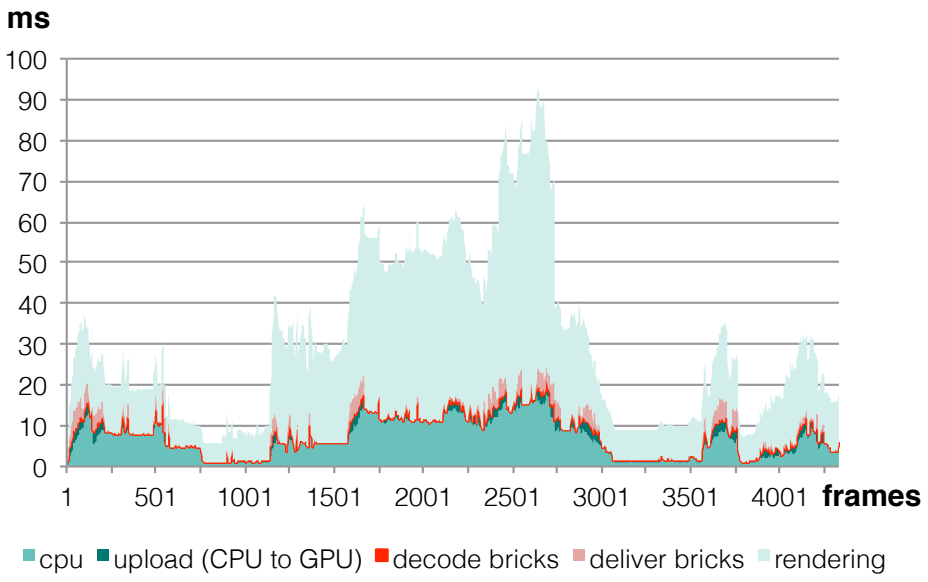
The number of rendered bricks per frame varies depending on the zoom factor, and is always maintained below 7000 by our adaptive renderer. Brick dequantization and reconstruction occurs only upon cache misses, which attributes to the low tensor reconstruction cost. But even under the most stressful situations where the number of rendered bricks changes rapidly, the dynamic update process is largely dominated by the brick data uploading time from CPU to GPU and not by the tensor reconstruction.

7.5.2 Application 2: Multiscale Feature Visualization

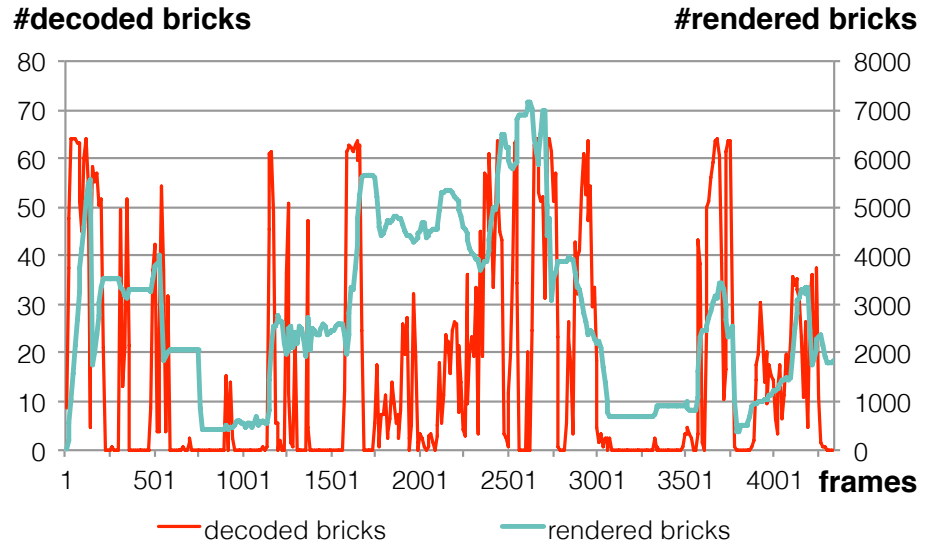
In this application, the implementation of the LOD error and rank coupling of multiresolution DVR and multiresolution feature visualization as published in [Suter et al., 2013] is presented. The application bases on the second multiresolution DVR model, as described in Sec. 7.3.

The goal of the interactive visualization system was to develop a multiresolution volume renderer that selects bricks not only at a certain spatial resolution, but also at a chosen feature scale. We achieved this by modeling the multiscalability within the multiresolution TA octree data structure, as shown in Sec. 7.3.3. The multiresolution renderer builds upon state-of-the-art view-dependent LOD selection, out-of-core data loading, brick caching on the GPU, asynchronous loading and rendering budgets. The bricks are decoded and reconstructed on demand using consecutive tensor times matrix multiplications, as introduced in [Suter et al., 2011], but using the new global mipmapped factor matrices hierarchy.

In addition to the view-dependent screen-projection based LOD selection, the feature scale of the reconstruction is chosen based on a user input (see Alg. 15).



(a) Performance in ms per frame



(b) Number of bricks per frame

Figure 7.15: *The performance measured is on the great ape molar: (a) indicates the per frame performance of different computing steps measured in ms, (b) shows the number of rendered and decoded bricks. Adapted from [Suter et al., 2011]*

The user input is a feature scale parameter, which maps to a tensor rank. A higher feature scale is achieved by reconstructing more ranks, a lower feature scale is achieved by reconstructing fewer ranks. During the actual visualization, a lower feature scale means that we perform a coarser approximation. This principle is exploited to steer the feature scale via the per-brick RMSE error ranges, as described in Sec. 7.3.3.

Algorithm 15 Per frame TA-error-based LOD traversal. From [Suter et al., 2013].

1: rendering list L, loading queue Q, heap H (screen-size sorting)
2: assign a rank corresponding to ε_{target} to the root brick $\mathscr{A}_{brick\_1}$
3: **if** $\mathscr{A}_{brick\_1}$ is on *GPU* and $\mathscr{A}_{brick\_1}$ is visible **then**
4: push $\mathscr{A}_{brick\_1}$ to H and then push $\mathscr{A}_{brick\_1}$ to L
5: **end if**
6: push $\mathscr{A}_{brick\_1}$ to Q
7: **while** H not empty **do**
8: set current brick $\mathscr{A}_{\text{brick}\downarrow k}$ to the front of H
9: remove front from H
10: **if** $(size(Q) \geq budget_{GPU})$ || $(size(L) \geq budget_{render})$ **then**
11: break
12: **else if** ($\mathscr{A}_{\text{brick}\downarrow k}$ has no children) || $(\varepsilon_{target} > \varepsilon_{rank=8}(\mathscr{A}_{\text{brick}\downarrow k}))$ || $[(\varepsilon_{target} > \varepsilon_{rank=32}(\mathscr{A}_{\text{brick}\downarrow k}))$ & $(screen\_size(\mathscr{A}_{\text{brick}\downarrow k}) < screen\_size\_threshold)]$ **then**
13: continue
14: **end if**
15: set list C to all visible children of $\mathscr{A}_{\text{brick}\downarrow k}$
16: assign all of C with ranks corresponding to ε_{target}
17: **if** all of C are on *GPU* **then**
18: sort C according to the rendering order
19: find $\mathscr{A}_{\text{brick}\downarrow k}$ in L and replace it with all of C
20: push all of C to H
21: **end if**
22: push all of C to Q
23: **end while**
24: update *GPU* usage statistics based on Q and L
25: request missing bricks on *GPU* from Q to (re)load async
26: render bricks from L

Fig. 7.16 illustrates the multiscale adjustments to the multiresolution LOD selection given a feature scale, represented by ε_{target} (Alg. 15, line 12). The minimum error front corresponds to $\varepsilon_{target} > \varepsilon_{rank=8}(\mathscr{A}_{\text{brick}\downarrow k})$, which prevents further resolution refinement; in contrast, the maximum error front corresponds to $\varepsilon_{target} > \varepsilon_{rank=32}(\mathscr{A}_{\text{brick}\downarrow k})$, which enforces refinement. During the brick refinement, the rank is updated based on the ε_{target} (Alg. 15, line 16). As mentioned in Alg. 15, line 10, we follow this adjusted multiresolution front as long as we stay within the given rendering and memory budgets.

In order to verify the multiscale and multiresolution tensor approximation hierarchy introduced in this paper, we implemented a volume rendering application

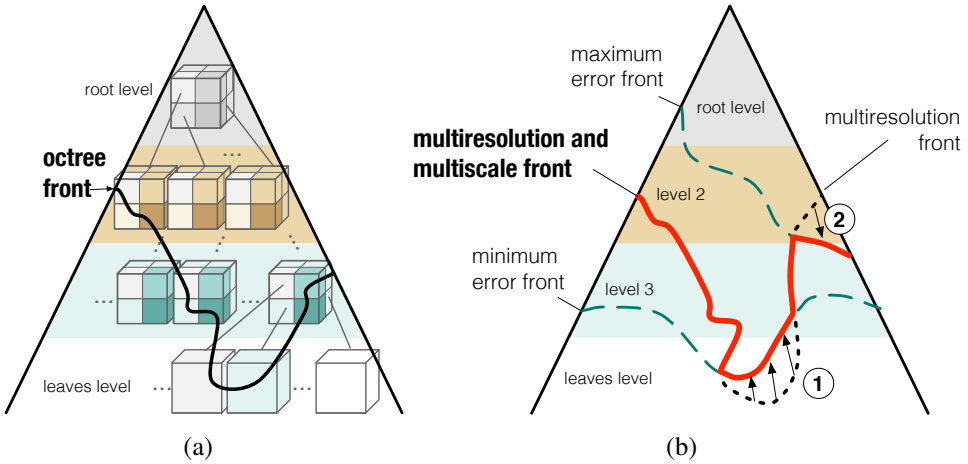
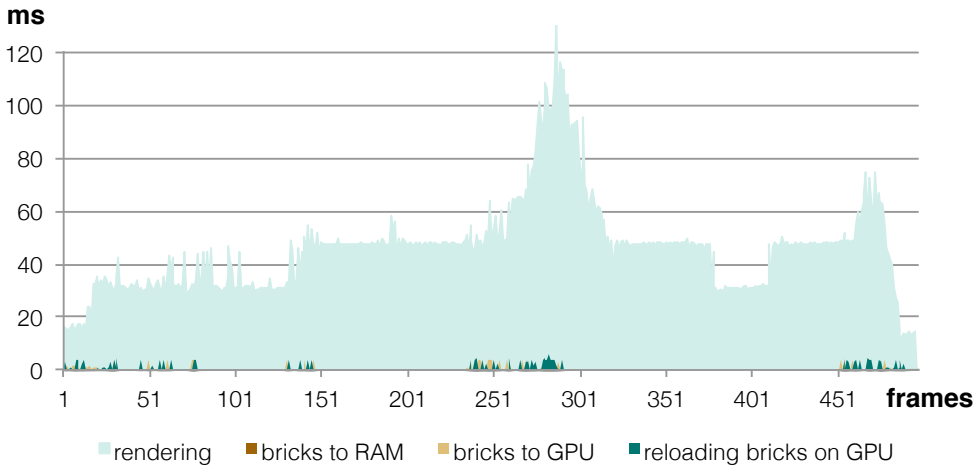


Figure 7.16: *(a) Illustration of a simplified multiresolution octree front. (b) Multiresolution and multiscale octree front (bold): The multiresolution front (dotted) is adjusted depending on (1) the minimum error octree front (prevent refinement), and (2) the maximum error octree front (enforce refinement).*

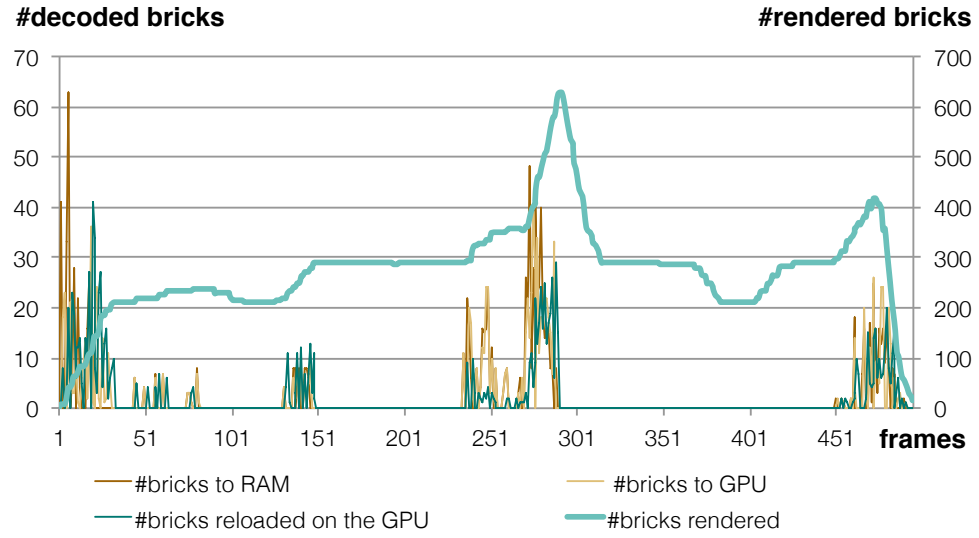
in C++ based on a GPU-ray caster that uses GLSL-shaders. The interactive visualization application is demonstrated on a Quad-Core Intel i7 3.2GHz with 8GB RAM and a Geforce GTX 580 with 1.5GB memory. The preprocessing has been carried out on an Quad-Core Intel Xeon 2.4GHz MacPro5,1 with 22GB RAM and a 500GB SSD hard disk, and a Geforce GTX 285 graphics card with 1GB memory. The test datasets include three μCT volumes: a *hazelnut* (512^3, $128MB$, $8bit$), a *flower* (1024^3, $1GB$, $8bit$) and a *wood branch* (2048^3, $16GB$, $16bit$) dataset (see App. A). To avoid excessive type conversions, the input data is preprocessed in floating point precision and the large data tensors (e.g. $32GB$ for wood branch) are accessed from memory mapped files.

As already shown in Sec. 7.5.1, the rendering from TA compressed data is dominated by the ray caster (see Fig. 7.17). Our multiresolution and multiscale DVR system shows interactive performance with an average frame rate (fps) of 25 and dropping to about 7 fps only briefly for the flower test data. The decompression and ray casting are performed in parallel in separate threads on the GPU. The rendering is performed adaptively according to the zooming factor and the defined error, which was chosen to be a medium error. The bricks are loaded according to the previously defined error-reconstruction-rank coupling.

In a multiresolution model, besides the interactive performance, the fit between the glued bricks is a quality measure. This is briefly analyzed next.



(a) Performance in ms per frame

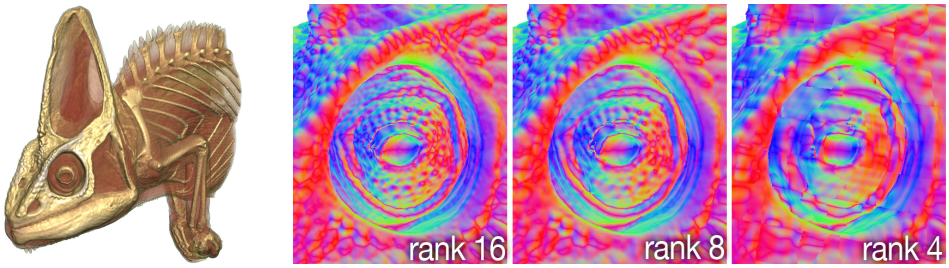


(b) Number of bricks per frame

Figure 7.17: *Performance measurements of the flower rendering. Time in ms per frame as well as number of loaded and rendered blocks per frame. From [Suter et al., 2013].*

7.5.3 Bricking and TA Multiresolution

The multiresolution DVR models base on bricking the input volume into subvolumes of equal sizes and of varying resolutions. During rendering the multiresolution bricks are glued together by the help of interpolating their gradients of brick border voxels. In the visualization system used for the bricked TA multiresolution model, it can be seen that even with a lower rank tensor approximation, i.e., by using less storage and bandwidth, the essential parts as well as details of a certain feature size can be visualized. Fig. 7.18(b) shows the effects of rank truncation on gradient quality. As we can see, block boundaries become apparent only at low ranks. Such artifacts are inherent to all brick-based lossy compression methods, and can be alleviated, at the cost of higher rendering time, by interblock interpolation through sampling neighboring bricks [Ljung et al., 2006a; Beyer et al., 2008] or by using deferred filtering approaches [Fout et al., 2005; Fout and Ma, 2007].



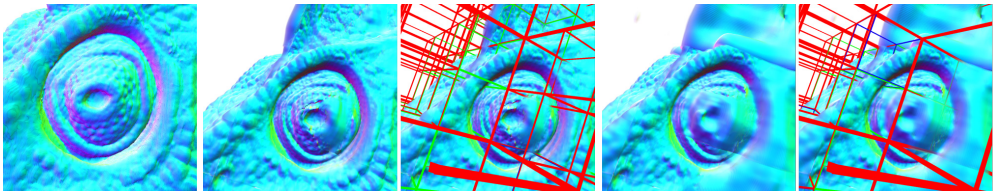
(a) veiled chameleon (b) gradients of bricking

Figure 7.18: *(a) Chameleon dataset rendered with the per brick multiresolution TA model; (b) comparison of various rank-(R,R,R) TAs using the gradient vector of the skin isosurface mapped to RGB colors, from [Suter et al., 2011].*

The global mipmapped multiresolution TA model shows more multiresolution glueing during rendering. This can be explained with the adaptive rank LOD error selection, which sometimes switches to lower resolutions if the error permits. Fig. 7.19 indicates that for a full resolution no brick artifacts are visible (left). For a medium and a higher error, artifacts are mainly visible due to different rank-reconstructions being selected for close neighboring bricks.

7.6 Summary

In this chapter, the four thesis-relevant implementation stages were elaborated. First, a template-based C++ implementation with all the relevant tensor approximation classes was developed and made available within the *vmmlib* open source



(a) full resolution (b) medium error (c) high error

Figure 7.19: *Mapping of gradient vector of the chameleon datasets' skin isosurface to RGB color: (a) Full resolution shows no bricking errors, (b-c) bricking effects for a medium and high LOD error, respectively. That is, multiple resolution and multiple ranks were merged into one screen image.*

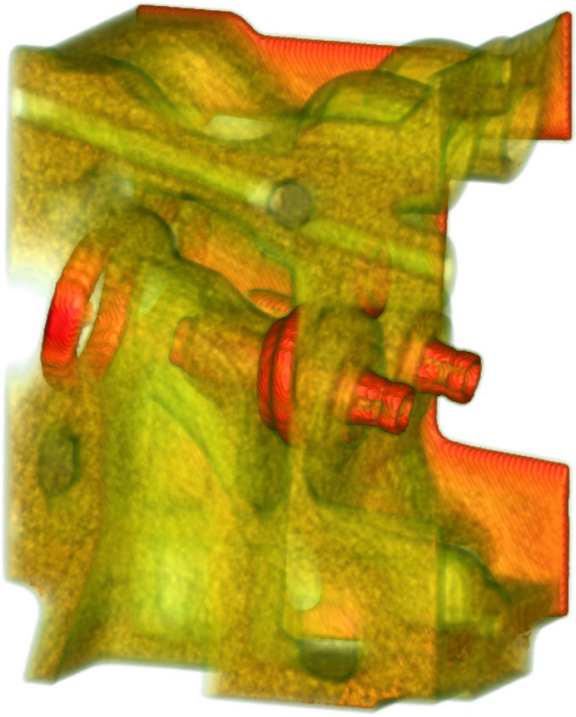
library [vmm, 2013]. This implementation includes memory-optimized and parallel executed tensor times matrix multiplications (TTMs), which were one of the bottlenecks during the tensor decomposition algorithm. Moreover, two different implementations for the higher-order SVD (HOSVD) were presented and analyzed. One of the approaches was based directly on the LAPACK SVD implementation (HOSVD), the other approach was based the symmetric eigenvalue decomposition from LAPACK (HOEIGS), which is applied on the covariance matrix of the unfolded input tensor, i.e., on $\mathbf{C}_{(n)} = \mathbf{A}_{(n)}\mathbf{A}_{(n)}^T$. As a conclusion, the tensor decomposition for smaller input tensors (e.g., octree bricks) the HOEIGS was faster, while for input tensors ($\geq \mathscr{A}_{512^3}$) the HOSVD was faster.

Second, a generic octree generation for the global TA bases multiresolution model was implemented. The implementation sequentially loads blocks of input data along the third spatial axis and produces the octree nodes. The lower resolution octree nodes are averaged on the fly, once the requested data is available. Each octree node is represented by a core tensor, matching to the data brick at the given octree node, and the corresponding submatrices of the global TA matrices. The core tensors are then stored in their quantized form (logarithmic encoding).

Third, the tensor reconstruction algorithm for interactive visualization was developed for GPU execution with CUDA. This implementation mainly consists of parallel TTM multiplications, which significantly reduces the computation time for the brick reconstructions. In fact, it was shown that the tensor reconstruction time was negligible compared to the overall rendering time, which is still the dominating factor. For both multiresolution models, interactive frame rates could be presented. Finally, multiresolution visualization (spatial axis in global TA bases) and the multiscalable feature representation (rank axis in the global TA bases) could successfully be coupled by the help of an error metric computed for all octree nodes and for different rank-reconstructions of the octree nodes.

C H A P T E R 8

CONCLUSIONS



3D direct volume visualization systems have become standard tools to analyze, explore and inspect large amounts of data. Direct volume visualization (DVR) approaches allow to visualize cross sections through a volume and to display transparent areas of an object rather than only an iso-surface as usually visualized by mesh approaches. However, DVR approaches are computationally more expensive. Moreover, 3D data acquisition devices are typically one step ahead of 3D visualization systems and produce datasets exceeding the graphics unit's memory. There are basically two strategies to address these bottlenecks:

- The actual amount of data can be reduced prior to rendering/visualization.
- The actual rendering efficiency can be optimized.

In DVR for large volumes, a key issue to address both these bottlenecks is an underlying multiresolution data structure, which organizes the input volume into an octree structure consisting of nodes of (downsampled) subvolumes at different resolutions. During rendering, the subvolumes can be loaded out-of-core at a certain resolution once requested. Today, it is state-of-the-art to perform the visualization directly on the graphics card. This makes the dataset size an even more precious parameter since the graphics memory is limited and data transfer from CPU to GPU during the interactive visualization is time consuming. Thus, it was a major thesis goal to reduce the size of the datasets as much as possible. A second thesis goal was to extract relevant features within a dataset during data reduction. Therefrom, the main thesis hypothesis was to find a unified mathematical framework for large volume visualization, which connects three tasks in one:

(a) Reduce the actual amount of data.

(b) Extract relevant features from the dataset.

(c) Visualize the data directly from the mathematical frameworks' coefficients.

In this thesis, tensor approximation (TA) was chosen as the unique framework since it is known, as a higher-order extension of PCA, to be a feature-sensitive approach. One strength of TA is its sensitivity on statistical properties like the major direction or periodicity. In contrast, state-of-the-art approaches like wavelet transform (WT) are rather beneficial when an overall statistical distribution of the dataset is intended to be reconstructed with fewer details and at a coarser resolution. Additionally, TA has shown to be effective at preserving actual three-dimensional features occurring in some of the sample data (e.g., dental growth structures). Based on this line of thoughts, TA was for the first time integrated into a multiresolution volume rendering system as part of this thesis. For this purpose, two different multiresolution TA models were developed and tested.

The first multiresolution TA model is a simple brick-based model, which encodes every octree node as a single tensor decomposition (bricked TA model). This basic TA model was further developed into a multiresolution model that consist of global mipmapped TA factor matrices (global TA bases model). Those global TA bases represent the basis for the whole octree and core tensors at every octree node capture the relationship of the original/downsampled data and the global TA factor matrices. Indeed, the global TA bases model exploits properties along the spatial dimension of the TA factor matrix bases. Spatial selectivity within the TA factor matrices is used for view-frustum culling and adaptive brick selection, spatial subsampling within the TA factor matrices is used for the downsampled lower-resolution representation of the full dataset.

Regarding data reduction, especially the global TA bases model reduced the storage costs significantly. The storage costs of the bricked TA model are roughly 20 percent of the original data elements ($0.2 \cdot I^3$), while the storage costs of the global TA bases model are reduced to about 15 percent of the original data elements ($0.15 \cdot I^3$). The coefficients of the tensor decomposition are in fact floating point numbers. Therefore, a tensor-specific quantization approach for the tensor coefficients was developed and constitutes another important contribution of this thesis. An 8-bit logarithmic quantization scheme for all core tensors and a 16-bit linear quantization have successfully been used for the TA factor matrices. In the global TA bases model, the factor matrices were, however, not encoded since they depict only a marginal data amount. In contrast, the TA factor matrices use a significant amount of storage in the bricked TA model. Specifically, incorporating the coefficient quantization, the storage costs for the bricked TA model raise to 25 percent of the input data elements ($0.25 \cdot I^3$).

Furthermore, the results of Chap. 6 indicate that the tensor rank can be used as a parameter to steer feature visualization at different scales (multiscalability). The tensor rank defines the number of column vectors and core tensor coefficients used for the reconstruction. Hence, the tensor rank is a parameter that adjusts (a) the amount of data used for the reconstruction, and (b) the scale of the features visualized in a certain reconstruction. Using more ranks adds details as well as finer scale features to a visualization, using only a few ranks visualizes the most prominent data structure (main statistical direction of the data distribution). Moreover, the multiscalability available through TA has been successfully coupled with the above mentioned multiresolution TA DVR models. That means the presented multiresolution TA DVR system presented in this thesis allows a researcher to visually explore features at multiple scales and at multiple resolutions.

Tensor approximation consists of two parts: (1) tensor decomposition, and (2) tensor reconstruction. In context with interactive out-of-core multiresolution DVR this parts received different attention during the implementation. The tensor decomposition was used during a non-time critical preprocessing routine to pro-

duce the TA multiresolution model. Therefore, CPU tensor classes with memory optimizations for large tensors were developed and contributed to the open-source library *vmmlib* [vmm, 2013]. Based on these tensor classes, a generic octree building procedure was developed. In contrast, the tensor reconstruction is critical to be performed in real-time. Thus, a parallel GPU-based tensor reconstruction was developed during this thesis. Interactive frame rates were successfully achieved. In fact, it could be shown that the imposed tensor reconstruction overhead is marginal compared to the overall rendering costs. The developed algorithms were tested on large volume datasets up to 32GB or 64GB when converted to floating point values, which are needed for the computations.

Finally, some of the acquired microCT datasets will be made publicly available in order to be used for comparison of algorithms in future work. This is an important issue, which allows to make research more comparable and permits other researchers to have a fast access to large volume test datasets.

The summarized thesis achievements and thesis contributions are illustrated in Fig. 8.1 in an updated visualization of Fig. 1.6.

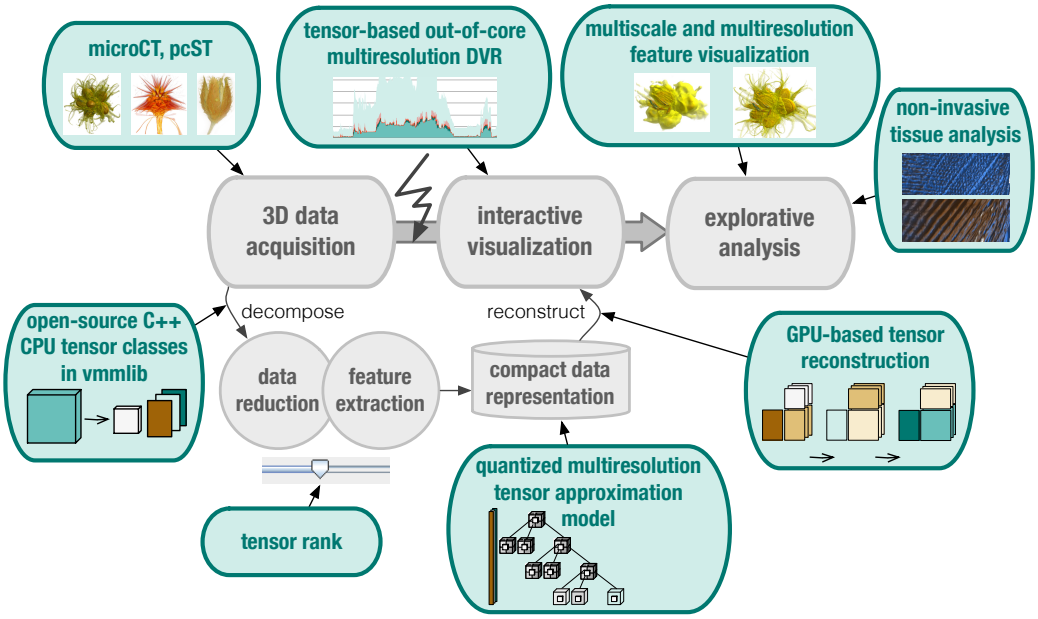


Figure 8.1: *The thesis achievements (green): A unified framework (tensor approximation) was successfully used to enable multiscale and multiresolution visualization to interactively explore features in large volume datasets.*

8.1 Future Work

The achievements and experiments regarding large volume data visualization performed during this thesis pointed out several future research directions and ideas, which are discussed here.

- *Experiments with Larger Volume Dataset Sizes*

  In this thesis, experiments with multiresolution tensor-based DVR systems were performed on datasets up to 32GB of input tensor size and up to 64GB of floating point tensor size needed for computing. This was a tensor of size $\mathscr{A}_{2048^2 \times 4096}$. For further experiments, new larger test volumes would need to be acquired. Another available test dataset would be the visible female dataset [US National Library of Medicine, 2003], which comprises a tensor $\mathscr{A}_{2048 \times 1216 \times 5189}$, i.e., roughly 50GB floating point dataset size. In fact, the visible female dataset is even available in different modalities (RGB of Cryosections, CT and MRI), which could maximally form a tensor $\mathscr{A}_{2048 \times 1216 \times 5189 \times 5}$. However, the MRI dataset is difficult to register with the other modalities since it was scanned separately and the scanning body setup hence differs too much for a meaningful registration. Thus, we could have a $\mathscr{A}_{2048 \times 1216 \times 5189 \times 4}$ of 200GB floating point dataset size used for computing, e.g., during the ALS. For this, the current TA DVR framework would need to be extended to higher orders.

- *Extension of Framework to Higher Orders*

  The direct volume visualization system presented in this thesis uses only third-order tensor approximation so far. However, tensor approximation itself is extendable to any number of higher-orders. Thus the decomposition and reconstruction algorithms as well as the visualization system are predestinated to be extended to deal with higher-order input tensors, e.g., 4th-order tensors. Here again, the visible female test dataset mentioned above is an ideal test case, where the fourth mode would be the data modalities. Other 4th-order input tensor applications would be any time-varying volume dataset, e.g., hurricane simulations.

- *Visualization of TA Derivatives*

  Another approach to extend and improve the available visualization features in a DVR system is to use gradients computed as central differences of the data elements. Gradients help to visually highlight depth and structures in a dataset. This approach could be easily implemented directly on the tensor factor matrices. Specifically, the central differences needed for the gradient computation can be computed along the factor matrices columns

corresponding to ranks. Therefore, this is a promising approach for tensor-based multiresolution visualization, in particular, for the global TA bases model presented in this thesis.

- *Encoding of Quantized TA Coefficients*

  Further storage cost reduction or data transmission reduction can be achieved by applying on the quantized tensor coefficients an encoding scheme such as a run-length encoding. The chosen method should allow for a fast random-access during reconstruction. However, since the tensor reconstruction step is marginal to the overall rendering costs, this aspect is not expected to carry significant weight on the tensor reconstruction process.

- *Sparse TA Representations*

  So far, only compact tensor approximation representations were used for the tensor-based multiresolution DVR. However, there is related work successfully employing sparse tensor representations, e.g., [Bader and Kolda, 2007; Liu et al., 2012; Chi and Kolda, 2012]. A suitable multiresolution sparse tensor approximation data structure could save even more storage costs and data transmission costs. Similar to the encoding of the quantized TA coefficients, one needs to verify that the time needed for reconstruction from a sparse multiresolution TA representation stays negligible.

- *Evaluation of Feature-sensitive Error Metrics*

  Finally, the experiments in Chap. 6 have shown that the applied error metrics were not always good indicators for extant features. Therefore, it would be worth to investigate and evaluate different error metrics that for any data reduction method measure both, the approximation quality of a dataset and the feature extraction quality. Examples of future investigations are given in [Wu et al., 2010; Lin and Kuo, 2011; Hill et al., 2011]. Such an evaluation could be further supported by a user study.

To sum up: In this thesis, tensor approximation was chosen as a unique framework for direct volume rendering of large datasets since it provides feature-sensitive bases. Moreover, TA provides one single parameter (the tensor rank), which enables an efficient data reduction and an effective feature extraction. Finally, the TA framework offers properties that fit multiresolution and multiscale volume rendering approaches.

APPENDIX A

DESCRIPTION OF DATASETS

A.1 Datasets from the Visualization Community

A.1.1 Bonsai

Description Bonsai tree

Resolution $\mathscr{A}_{256\times256\times128}$

Voxel size $0.585938mm \times 0.585938mm \times 1mm$

Voxel format unsigned 8-bit

Acquisition modality CT (contrast dye)

Source Christof Rezk-Salama, University of Erlangen-Nuremberg, Germany

A.1.2 Bonsai256

Description Bonsai tree

Resolution $\mathscr{A}_{256\times256\times256}$

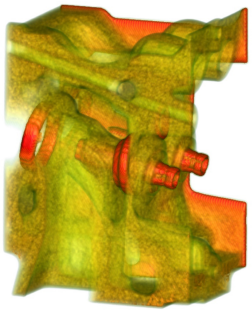
Voxel size $1 \times 1 \times 1$

Voxel format unsigned 8-bit

Acquisition modality CT

Source Stefan Roettger, University of Stuttgart, Germany
`http://www.volvis.org`

A.1.3 Engine



Description Two cylinders of an engine block

Resolution $\mathscr{A}_{256\times256\times128}$

Voxel size $1 \times 1 \times 1$

Voxel format unsigned 8-bit

Acquisition modality CT

Source General Electric, `http://www.volvis.org`

A.1.4 Veiled Chameleon

Description Veiled chameleon, field of reconstruction is 94.5*mm*

Resolution $\mathscr{A}_{1024\times1024\times1080}$

Voxel size $1 \times 1 \times 1$

Voxel format unsigned 16-bit

Acquisition modality CT

Source Digital Morphology Project, University of Texas, Austin, USA
`http://www.digimorph.org/specimens/Chamaeleo_calyptratus/whole/`

A.1.5 VIX

Description Feet

Resolution $\mathscr{A}_{512\times512\times250}$

Voxel size $0.40234375mm \times 0.40234375mm \times 1mm$

Voxel format unsigned 16-bit

Acquisition modality CT

Source OsiriX, `http://www.osirix-viewer.com/datasets`

A.2 Acquired Datasets

A.2.1 Hazelnut (Hnut)

Description Dried Hazelnut

Resolution $\mathscr{A}_{512\times512\times512}$

Voxel size $148\mu m \times 148\mu m \times 148\mu m$

Voxel format unsigned 8-bit

Acquisition modality microCT

Source Susanne Suter, University of Zurich, Switzerland

A.2.2 Beechnut

Description Dried beechnut

Resolution $\mathscr{A}_{1024\times1024\times1546}$

Voxel size $20\mu m \times 20\mu m \times 20\mu m$

Voxel format unsigned 16-bit

Acquisition modality microCT

Source Susanne Suter, University of Zurich, Switzerland

A.2.3 Flower (Leucadendron rubrum)

Description Dried flower (Leucadendron rubrum)

Resolution $\mathscr{A}_{512\times512\times512}$

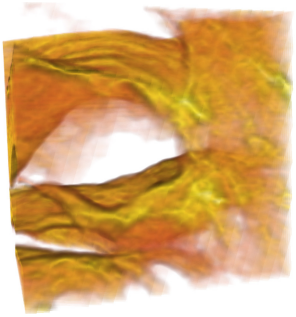
Voxel size $60\mu m \times 60\mu m \times 60\mu m$

Voxel format unsigned 8-bit

Acquisition modality microCT

Source Susanne Suter, University of Zurich, Switzerland

A.2.4 Brick64 (Flower Subvolume)



Description Subvolume of dried flower scan $\mathscr{A}(600:663, 512:576, 512:576)$

Resolution $\mathscr{A}_{64\times64\times64}$

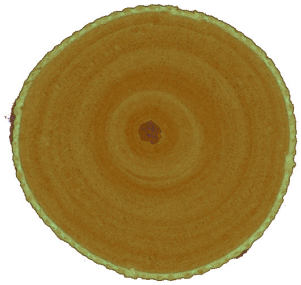
Voxel size $60\mu m \times 60\mu m \times 60\mu m$

Voxel format unsigned 8-bit

Acquisition modality microCT

Source Susanne Suter, University of Zurich, Switzerland

A.2.5 Wood Branch



Description Wood branch (of a hazelnut tree)

Resolution $\mathscr{A}_{2048\times2048\times4096}$

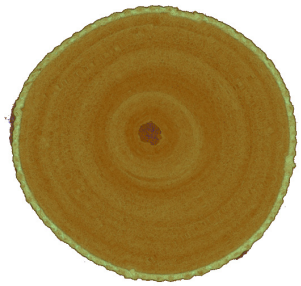
Voxel size $18\mu m \times 18\mu m \times 18\mu m$

Voxel format unsigned 16-bit

Acquisition modality microCT

Source Susanne Suter, University of Zurich, Switzerland

A.2.6 Subvolume of Wood Branch



Description Wood branch (of a hazelnut tree)

Resolution $\mathscr{A}_{2048\times2048\times2048}$

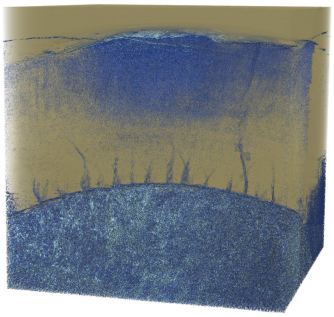
Voxel size $18\mu m \times 18\mu m \times 18\mu m$

Voxel format unsigned 16-bit

Acquisition modality microCT

Source Susanne Suter, University of Zurich, Switzerland

A.2.7 Great Ape Molar



Description Great ape molar, $7mm^3$

Resolution $\mathscr{A}_{2048\times2048\times2048}$

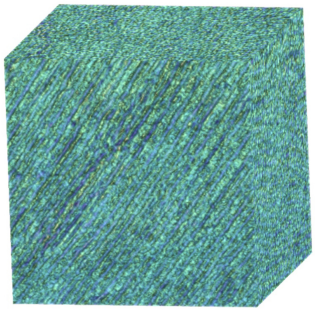
Voxel size $3.5\mu m \times 3.5\mu m \times 3.5\mu m$

Voxel format unsigned 16-bit

Acquisition modality PCST (Swiss Light Source)

Source Christoph Zollikofer, University of Zurich, Switzerland

A.2.8 Great Ape Molar Subvolume



Description A great ape molar subvolume (Sec. A.2.7)

Resolution $\mathscr{A}_{256\times256\times128}$

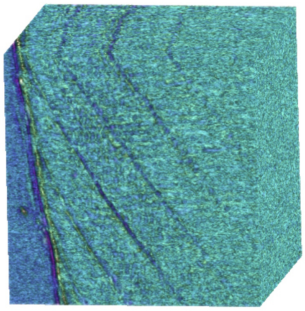
Voxel size $3.5\mu m \times 3.5\mu m \times 3.5\mu m$

Voxel format unsigned 16-bit

Acquisition modality PCST (Swiss Light Source)

Source Christoph Zollikofer, University of Zurich, Switzerland

A.2.9 Dental Subvolume



Description A great ape molar subvolume showing dental growth patterns

Resolution $\mathscr{A}_{256\times256\times256}$

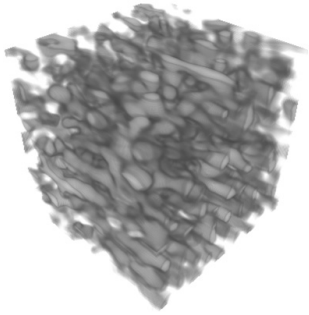
Voxel size $1 \times 1 \times 1$

Voxel format unsigned 16-bit

Acquisition modality PCST (Swiss Light Source)

Source Christoph Zollikofer, University of Zurich, Switzerland

A.2.10 Dental Subvolume



Description A great ape molar subvolume showing dental growth patterns

Resolution $\mathscr{A}_{64\times64\times64}$

Voxel size $1 \times 1 \times 1$

Voxel format unsigned 16-bit

Acquisition modality PCST (European Synchrotron Radiation Facility)

Source Christoph Zollikofer, University of Zurich, Switzerland

APPENDIX B

LINEAR ALGEBRA BACKGROUND

In this appendix, some thesis-relevant linear algebra background is given. In this thesis, we work with so-called tensor approximations, which extend the concept of the singular value decomposition (SVD) to higher orders. Therefore, the underlying SVD algorithm and the constraints and properties of the decomposition are summarized in what follows. The SVD is one of the most widely used matrix factorization method. Another closely related matrix factorization method is the eigenvalue decomposition, which can under certain constraints replace the SVD. The relationship is elaborated in this appendix on the basis of real values. Background literature that served as a basis to produce this summary can be found in these works [Trefethen and Bau, 1997; Strang, 1998; Madsen et al., 2004; Persson, 2007; Strang, 2007; Moler, 2008; Strang, 2009; Davis, 2012].

B.1 Eigenvalues and Eigenvectors

Eigensystems are most important in modeling dynamic problems (change over time, growth, decrease, oscillation), which are not solvable by elimination approaches. The *eigenvalue decomposition* (EIG) is used to factorize a square matrix $\mathbf{A} \in \mathbb{R}^{M \times M}$ into a non-singular matrix $\mathbf{S} \in \mathbb{R}^{M \times M}$ and a diagonal matrix $\Lambda \in \mathbb{R}^{M \times M}$ as shown in Eq. (B.1).

$$\mathbf{A} = \mathbf{S}\Lambda\mathbf{S}^{-1} \tag{B.1}$$

The columns of the matrix $\mathbf{S}$ are the *eigenvectors* $\boldsymbol{x}_j$, while the diagonal values

of the matrix Λ are the *eigenvalues* λ_j. Denoting the j-th column of $\mathbf{S}$ with $\boldsymbol{x}_j$, it can be easily shown that $\mathbf{A}\boldsymbol{x}_j = \lambda_j \boldsymbol{x}_j$. The intuition behind an eigenvector is to find a non-zero vector $\boldsymbol{x}_j$, which has the same direction as $\mathbf{A}\boldsymbol{x}_j$. The scaling number λ_j determines how the vector $\boldsymbol{x}_j$ is transformed by $\mathbf{A}$, i.e., whether $\boldsymbol{x}_j$ is unchanged ($\lambda == 1$), compressed ($\lambda < 1$), stretched ($\lambda > 1$) or mirrored ($\lambda < 0$). Finally, an eigenvalue decomposition can be seen as a change of basis to "eigenvalue coordinates".

Note: The number λ can be a multiple eigenvalue, meaning that multiple eigenvectors can correspond to one single eigenvalue. The dimension of the eigenspace E_λ can be interpreted as the maximum number of linearly independent eigenvectors that can be found, all with the same eigenvalue λ (geometric multiplicity, see also algebraic multiplicity). Therefore, the eigenvectors are often divided by their length in order to receive one unit eigenvector. In practice, for most matrices the number of eigenvalues and eigenvectors is equal to the rank of the matrix. The eigenvalue properties can be summarized as:

Eigenvalue Properties

- A number λ is an eigenvalue of the matrix $\mathbf{A}$ if $det(\mathbf{A} - \lambda\mathbf{I}) = 0$, where $\mathbf{I}$ is the identity matrix.
- The eigenvalues of $\mathbf{A}^2$ and $\mathbf{A}^{-1}$ are λ^2 and λ^{-1}, respectively.
- The product of the M eigenvalues of the matrix $\mathbf{A}$ equals the determinant of the matrix $\mathbf{A}$: $\lambda_1 \cdot \lambda_2 \cdot \cdots \cdot \lambda_M = det(\mathbf{A})$.
- The sum of the M eigenvalues is equal to the sum of the n diagonal entries of the matrix $\mathbf{A}$, also-called the trace of the matrix $\mathbf{A}$: $\lambda_1 + \lambda_2 + \cdots + \lambda_M = trace(\mathbf{A}) = a_{11} + a_{22} + \cdots + a_{mm}$.

Special characteristics of the eigenvalue decomposition hold for symmetric matrices ($\mathbf{A} = \mathbf{A}^T$):

Eigenvalue Properties of Symmetric Matrices

- The eigenvalues of a symmetric matrix are real values.
- The eigenvectors of a symmetric matrix can be chosen to be orthogonal.
- All symmetric matrices are diagonalizable (*spectral theorem*).

Hence for symmetric matrices the eigenvalue decomposition from Eq. (B.1) can be written as in Eq. (B.2), where $\mathbf{Q}$ is an orthogonal or orthonormal and square matrix $\mathbf{Q}^T = \mathbf{Q}^{-1}$. Orthonormal vectors are convenient since they never overflow or underflow.

$$\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^{-1} = \mathbf{Q}\Lambda\mathbf{Q}^T \tag{B.2}$$

While the eigenvalue decomposition only works for square matrices, the singular value decomposition or short SVD is applicable to any rectangular matrix.

B.2 The Singular Value Decomposition (SVD)

The singular value decomposition (SVD) is a widely used matrix factorization procedure to solve linear least-square problems. The SVD can be applied to any square or rectangular matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$. Hence, the decomposition is always possible. The aim of the SVD is to produce a diagonalization of the input matrix $\mathbf{A}$. Since the input matrix $\mathbf{A}$ is not symmetric, two bases (matrices) are needed to diagonalize $\mathbf{A}$. Therefore, the SVD produces a matrix factorization into two orthogonal bases $\mathbf{U} \in \mathbb{R}^{M\times M}$ and $\mathbf{V} \in \mathbb{R}^{N\times N}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{M\times N}$, as expressed in Eq. (B.3) (matrix form) or Eq. (B.4) (summation form).

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{-\mathbf{1}} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}} \tag{B.3}$$

$$a_{mn} = \sum_{r=1}^{P} u_{mp}\sigma_p v_{np} \tag{B.4}$$

The bases $\mathbf{U}$ and $\mathbf{V}$ contain orthogonal unit length vectors $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$, respectively, and represent a r-dimensional column space ($\mathbb{R}^M$) and a r-dimensional row space ($\mathbb{R}^N$). Hence, the bases $\mathbf{U}$ and $\mathbf{V}$ are even orthonormal, as indicated in Eq. (B.3), where the inverse of the matrix $\mathbf{V}^{-1}$ equals its transpose $\mathbf{V}^{\mathbf{T}}$. The diagonal matrix Σ contains the *singular values* σ_i, where $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_P \geq 0$, where $P = \min(M,N)$. The number of non-zero singular values determines the rank R of the matrix $\mathbf{A}$. The SVD can be seen as linear transformation of the orthogonal vectors $\boldsymbol{u}_j$ into the orthogonal vectors $\boldsymbol{v}_j$, where σ_j is the scaling factor. That is: $\mathbf{A}\cdot\boldsymbol{v}_j = \sigma_j \cdot \boldsymbol{u}_j$ or $\mathbf{AV} = \mathbf{U}\Sigma$ for the full decomposition. In some applications truncated versions of the SVD are desired. That is, only the first K singular values $\sigma_1 \ldots \sigma_K$ and the corresponding K singular vectors $\boldsymbol{u}_1 \ldots \boldsymbol{u}_K$ and $\boldsymbol{v}_1 \ldots \boldsymbol{v}_K$ are used for the reconstruction. This approach is referred to as low-rank approximation of a truncated SVD. The most important properties of the matrix SVD can be summarized as:

Matrix SVD Properties

- The matrices $\mathbf{U}$ and $\mathbf{V}$ are both orthonormal, meaning that their columns $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ are orthonormal; $\mathbf{V}$ is even row-orthonormal, since it is a square matrix.
- The scaling factors (or singular values) σ_j are arranged in decreasing order of magnitude: $\sigma_1 \geq \cdots \geq \sigma_P \geq 0$, where $P = \min(M,N)$.
- The SVD is a rank-R decomposition where the number of non-zero singular values indicates the matrix rank $R = rank(\mathbf{A})$.

B.2.1 Computing the SVD

Most frequently, the SVD is computed by using a Householder reduction to a bidiagonal matrix followed by a diagonalization using the QR factorization (for details we refer to [Press et al., 1992; Golub and Van Loan, 1996]). However, the SVD can also be computed by using symmetric eigenvalue decomposition. That means, instead of computing the SVD of $\mathbf{A}$, we compute the symmetric eigenvalue decomposition of $\mathbf{A}\mathbf{A}^T$ or $\mathbf{A}^T\mathbf{A}$, which are both symmetric matrices and referred to as covariances matrices of $\mathbf{A}$. In order to find the $\boldsymbol{u}_1 \ldots \boldsymbol{u}_m$, we use the symmetric matrix $\mathbf{A}\mathbf{A}^T$ (Eq. (B.5)); in order to find the $\boldsymbol{v}_1 \ldots \boldsymbol{v}_n$, we produce the symmetric matrix $\mathbf{A}^T\mathbf{A}$ and decompose it as in Eq. (B.6). P is the number of singular values, where $P = \min(M,N)$.

$$\mathbf{A}\mathbf{A}^T = (\mathbf{U}\Sigma\mathbf{V}^\mathbf{T})(\mathbf{U}\Sigma\mathbf{V}^\mathbf{T})^T = \mathbf{U}\Sigma^T\mathbf{V}^T\mathbf{V}\Sigma\mathbf{U}^\mathbf{T} = \mathbf{U}\begin{bmatrix}\sigma_1^2 & & \\ & \ddots & \\ & & \sigma_P^2\end{bmatrix}\mathbf{U}^\mathbf{T} \tag{B.5}$$

$$\mathbf{A}^T\mathbf{A} = (\mathbf{U}\Sigma\mathbf{V}^\mathbf{T})^T(\mathbf{U}\Sigma\mathbf{V}^\mathbf{T}) = \mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^\mathbf{T} = \mathbf{V}\begin{bmatrix}\sigma_1^2 & & \\ & \ddots & \\ & & \sigma_P^2\end{bmatrix}\mathbf{V}^\mathbf{T} \tag{B.6}$$

Note that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{U}^T = \mathbf{U}^{-1}$, $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and $\mathbf{V}^T = \mathbf{V}^{-1}$. Thus in the example of the matrix $\mathbf{V}$ computation, $\mathbf{V}\begin{bmatrix}\sigma_1^2 & & \\ & \ddots & \\ & & \sigma_P^2\end{bmatrix}\mathbf{V}^\mathbf{T}$ has the same form as an eigenvalue decomposition of a symmetric matrix (Eq. (B.2)), where the

symmetric matrix is $\mathbf{A}^T\mathbf{A}$. The columns of $\mathbf{V}$ are the eigenvectors of this matrix. The diagonal matrix produces the squares σ^2 of the singular values σ. Note, no matter with which initial symmetric covariance matrix ($\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$) we start, the non-zero eigenvalues stay the same.

B.2.2 Full vs. Reduced SVD

The singular value decomposition is usually represented in its compact or reduced form (Fig. B.1(b)). If we look at the full SVD in Fig. B.1(a), we notice that there are only P singular values, where $P = \min(M,N)$, in the diagonal matrix Σ. Therefore, the last columns of $\mathbf{U}$ will be multiplied by zeros. Hence, it is more economic to use the reduced form for computations using the SVD. For so-called low-rank approximations, even smaller decompositions are required known as partial or truncated SVD (Fig. B.1(c)) and limiting the number of singular values to $K < P$. In other words, the full SVD has P singular values, the compact/reduced SVD has N singular value and the partial/truncated SVD has K singular values.

B.3 Eigenvalues vs. Singular Values

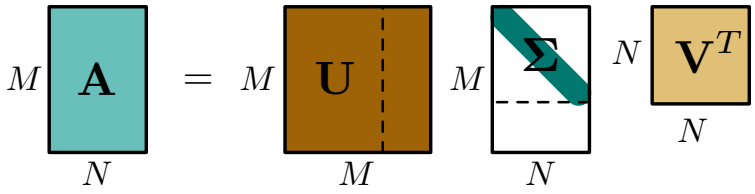
In this section, differences and commonalities between the eigenvalues and singular values and the eigenvectors and singular vectors are elaborated. The definitions are taken from [Moler, 2008; Persson, 2007; Madsen et al., 2004].

- An *eigenvalue* and *eigenvector* of a square matrix $\mathbf{A}$ are a scalar λ and a non-zero vector $\boldsymbol{x}_j$ such that $\mathbf{A}\boldsymbol{x}_j = \lambda_j\boldsymbol{x}_j$. An eigenvalue (from German "Eigenwert") represents an "own value" or a "characteristic value".

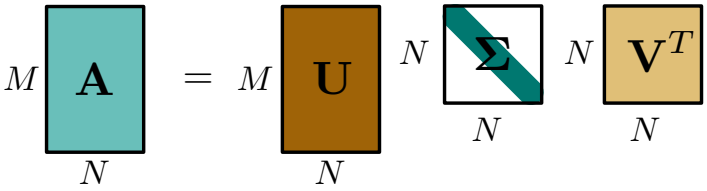
- A *singular value* and a pair of *singular vectors* of a square or rectangular matrix $\mathbf{A}$ are a non-negative scalar σ and two non-zero vectors $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ so that $\mathbf{A}\boldsymbol{v}_j = \sigma_j\boldsymbol{u}_j$ or $\mathbf{A}^T\boldsymbol{u}_j = \sigma_j\boldsymbol{v}_j$. The vectors $\boldsymbol{u}_j$ are the *left singular vectors*, and the vectors $\boldsymbol{v}_j$ are the *right singular vectors*.

- Eigenvalues are used for systems where the matrix is a transformation from a vector space to itself. On the other hand, singular values are used when the matrix is transformed from one vector space to a different vector space.

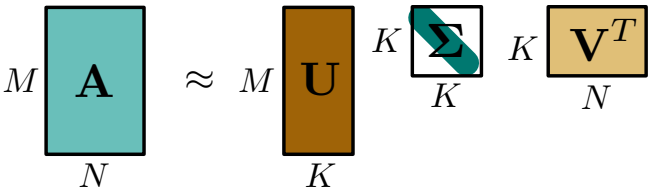
- An eigenvector $\boldsymbol{x}_j$, or a pair of singular vectors $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$, can be scaled by any non-zero factor without changing any important properties. Eigenvectors of symmetric matrices are usually normalized to have Euclidean length equal to one, $\left\|x_j\right\|_2 = 1$.



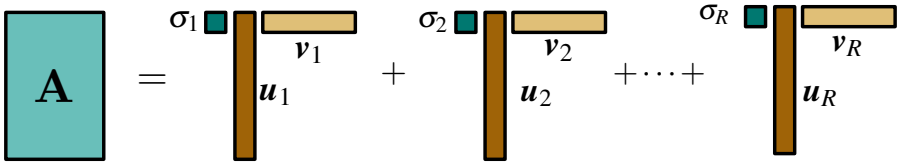
(a) full SVD



(b) reduced SVD



(c) truncated SVD



(d) summed SVD

Figure B.1: *SVD variants: (a) full SVD (P singular values, where $P = \min(M,N)$), (b) reduced/compact SVD (N singular values), and (c) truncated/partial SVD (K singular values). (d)Visualization of the summed form of the SVD as shown in Eq.* (B.4).

- The *eigenvalue decomposition* (Eq. (B.1)) uses the same basis $\mathbf{S}$ for row and column space, but the SVD (Eq. (B.3)) uses two different bases $\mathbf{V}$, $\mathbf{U}$. Moreover, the eigenvalue decomposition generally does not use an orthonormal basis, but the SVD does. The eigenvalue decomposition is only defined for square matrices, but the SVD exists for all matrices.

- For *symmetric positive definite* matrices $\mathbf{A}$, the eigenvalue decomposition and the SVD are equal.

- The *singular values* of the square matrix $\mathbf{A}$ are defined as the square roots of the eigenvalues of $\mathbf{A}^T\mathbf{A}$ or $\mathbf{A}\mathbf{A}^T$.

This summarizes the most important facts about the matrix SVD. The matrix SVD extension to higher orders is not unique, but is summarized under the term *tensor approximation*, as introduced in Sec. 3.3.

APPENDIX C

COMPUTING WITH TENSORS

Here, the most common products used while computing with tensors are outlined. The notation taken here is mostly taken from [Kolda and Bader, 2009] and follows the notations proposed by Kiers [Kiers, 2000]. Some notations are, however, taken from [De Lathauwer, 2009] and [Smilde et al., 2004].

- An N^{th}-order tensor is defined as $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$.
- The *tensor product* is denoted here by $\otimes$: however, other symbols are used in the literature, too. For rank-one tensors, the tensor product corresponds to the *vector outer product* ($\circ$) of N vectors $\boldsymbol{b}^{(n)} \in \mathbb{R}^{I_n}$ and results in an N^{th}-order tensor $\mathscr{A}$. The tensor product or vector outer product for a 3^{rd}-order rank-one tensor is illustrated in Fig. C.1: $\mathscr{A} = \boldsymbol{b}^{(1)} \circ \boldsymbol{b}^{(2)} \circ \boldsymbol{b}^{(3)}$, where an element (i_1, i_2, i_3) of $\mathscr{A}$ is $a_{i_1 i_2 i_3} = b^{(1)}_{i_1} b^{(2)}_{i_1} b^{(3)}_{i_3}$.

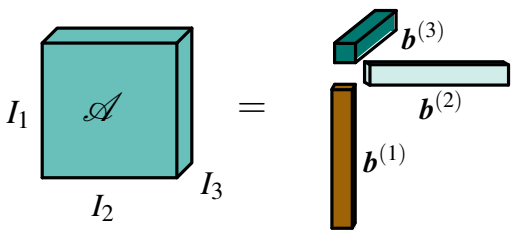


Figure C.1: *Three-way outer product for a rank-one tensor3* $\mathscr{A} = \mathbf{b}^{(1)} \circ \mathbf{b}^{(2)} \circ \mathbf{b}^{(3)}$.

- The *inner product* of two same-sized tensors $\mathscr{A}, \mathscr{B} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the sum of the products of their entries, i.e., Eq. (C.1).

$$(\mathscr{A}, \mathscr{B}) = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} a_{i_1,i_2,\ldots i_N} b_{i_1,i_2,\ldots i_N} \tag{C.1}$$

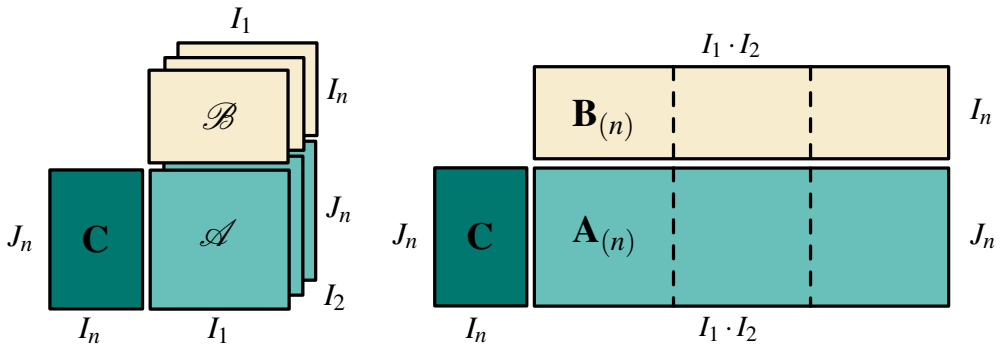
- The *n-mode product* [De Lathauwer et al., 2000a] multiplies a tensor by a matrix (or vector) in mode n. The n-mode product of a tensor $\mathscr{B} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{C} \in \mathbb{R}^{J_n \times I_n}$ is denoted by $\mathscr{B} \times_n \mathbf{C}$ and is of size $I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N$. That is, element-wise we have Eq. (C.2).

$$(\mathscr{B} \times_n \mathbf{C})_{i_1 \ldots i_{n-1} j_n i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} b_{i_1 i_2 \ldots i_N} \cdot c_{j_n i_n} \tag{C.2}$$

  Each mode-n fiber is multiplied by the matrix $\mathbf{C}$. The idea can also be expressed in terms of unfolded tensors (reorganization of tensor into a matrix; see Sec. 3.3.1).

$$\mathscr{A} = \mathscr{B} \times_n \mathbf{C} \Leftrightarrow \mathbf{A}_{(n)} = \mathbf{C}\mathbf{B}_{(n)} \tag{C.3}$$

  The n-mode product of a tensor with a matrix is related to a change of basis in the case when a tensor defines a multilinear operator [Kolda and Bader, 2009]. The n-mode product is the generalized operand to compute tensor times matrix (TTM) multiplications, as illustrated in Fig. C.2.



(a) TTM of a tensor3 (b) TTM of an unfolded tensor3

Figure C.2: *Tensor times matrix (TTM) multiplication.*

- The *Hadamard product* ($*$) is the element-wise product between two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{I \times J}$ of the same size (see Eq. (C.4)).

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & \dots & a_{1J}b_{1J} \\ \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & \dots & a_{IJ}b_{IJ} \end{bmatrix} \tag{C.4}$$

- The *Kronecker product* ($\otimes$) multiplies two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times M}$ block-wise as in Eq. (C.5), while the resulting matrix $\mathbf{A} \otimes \mathbf{B}$ is of size $(IK \times JM)$. The Kronecker product ($\otimes$) is denoted by the same operator as the outer product and is a generalization of the vector outer product to matrices. The Kronecker product is in fact a special case of the tensor product, but not every tensor product is a Kronecker product [Burdick, 1995].

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1J}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & \dots & a_{IJ}\mathbf{B} \end{bmatrix} \tag{C.5}$$

- The *Khatri-Rao product* ($\odot$) [Smilde et al., 2004] is denoted as a column-wise Kronecker product. The resulting matrix $\mathbf{A} \odot \mathbf{B}$ is of size $(IJ) \times K$ for the two matrices $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$ (see Eq. (C.6)).

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \dots & a_K \otimes b_K \end{bmatrix} \tag{C.6}$$

  Note: If $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors, then the Khatri-Rao and Kronecker products are identical, i.e., $\boldsymbol{a} \otimes \boldsymbol{b} = \boldsymbol{a} \odot \boldsymbol{b}$.

- The *Moore-Penrose inverse* [Moore, 1920; Penrose, 1955] is a generalized matrix pseudo inverse $\mathbf{A}^+ \in \mathbb{R}^{I \times J}$, which works for rectangular matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$. There are other matrix pseudo inverses; however, in this thesis the robust and SVD-based Moor-Penrose inverse is used: $\mathbf{A}^+ = \mathbf{U}\Sigma^+\mathbf{V}^T$, where Σ^+ represents the pseudo inverse of Σ as in Eq. (B.3) of the SVD.

- The *norm of a tensor* $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ is defined analogously to the matrix Frobenius norm $\|\mathbf{A}\|_F$ and is the square root of the sum squares of all its elements, i.e., Eq. (C.7).

$$\|\mathscr{A}\|_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \dots \sum_{i_N=1}^{I_N} a_{i_1,i_2,\dots i_N}^2} \tag{C.7}$$

APPENDIX D

TENSOR DECOMPOSITION ALGORITHMS

There are a couple of different strategies for how to perform tensor decompositions or rank approximations. The most popular and widely used group of algorithms belongs to the *alternating least squares* (ALS) algorithms. The other group of algorithms use various Newton methods. The respective algorithms differ also for the computation of the CP model and the Tucker model.

For the Tucker model, the first decomposition algorithms used were a simple HOSVD, the so-called *Tucker1* [Tucker, 1966], the three-mode SVD, and the HOSVD, the higher-order generalization [De Lathauwer et al., 2000a] that was described in Sec. 3.3.2. However, the truncated decompositions of the HOSVD are not optimal in terms of best fit, which is measured by the Frobenius norm of the difference. Starting from an HOSVD, tensor approximation ALS algorithms [Kroonenberg and De Leeuw, 1980; Kroonenberg, 1983] were developed, where one of the first Tucker ALS was the so-called *TUCKALS* [Ten Berge et al., 1987]. Later various optimizations accelerated [Andersson and Bro, 1998] or optimized the basic TUCKALS. The *higher-order orthogonal iteration* (HOOI) algorithm [De Lathauwer et al., 2000b] is an iterative algorithm to perform a better fit for a truncated HOSVD version.

Newton methods are used for the Tucker decomposition or rank-$(R_1, R_2, \ldots, R_N)$ approximation as well. They typically start with an HOOI initialization and then converge faster to the final point. [Elden and Savas, 2009] developed a *Newton-*

Grassman optimization approach, which takes much fewer iterations than the HOOI - even though one single iteration is more expensive due to the computation of the Hessian. While the HOOI is not guaranteed to converge, the Newton-Grassmann Tucker decomposition is guaranteed to converge to a stationary point. Another Newton method was proposed by [Ishteva et al., 2009], who developed a *differential-geometric Newton* algorithm with a fast quadratic convergence of the algorithm in a neighborhood of the solution. Since this method is not guaranteed to converge to a global maximum, they support the method by starting with an initial guess of several HOOI iterations, which increases the chances of converging to a solution.

For the CP model, one question addressed is how to find the number of rank-one tensors: CORCONDIA [Bro and Kiers, 2003] is an algorithm that performs a consistency diagnostic to compare different numbers of components. For a fixed number of components, there is a CP ALS algorithm, which was presented in the two original CP articles [Caroll and Chang, 1970; Harshman, 1970]. [Zhang and Golub, 2001] proposed to use *incremental rank-one fitting procedures*, which first fit the original tensor by a rank-one tensor, then subtract the rank-one approximation from the original and fit the residue with another rank-one tensor until a certain given number of F incremental rank-one approximations have been performed. They propose a *Jacobi Gauss-Newton* (JGN) iteration to execute the incremental rank-one approximations.

ALS Algorithms Alternating least-squares algorithms are used to find parameters of a model, which corresponds to an optimization problem. In particular, if no closed-form solutions to problems are available, iterative algorithms that gradually improve the estimates and converge to the optimum solution are used. The tensor ALS produces a tensor decomposition consisting of N basis matrices $\mathbf{U}^{(1\ldots N)}$ and coefficients representing the relationship between the input tensor and the basis matrices (see Sec. 3.4). The general idea with the multiway/tensor ALS algorithms is to fix all basis matrices but one and optimize only for $\mathbf{U}^{(n)}$. By fixing all bases but one, the optimization problem is reduced to a linear least squares problem. This procedure is repeated for every mode-n basis. One iteration step comprises the optimization of all bases individually. The improvement of the solution is measured after each iteration by a predefined *set of convergence/stopping criteria*, which decides if the current fit is considered to be the best fit.

Often it is difficult to define the stopping criteria [Kroonenberg, 2008]. In order to have a termination of the algorithm, a maximum number of iterations should be set since ALS algorithms typically suffer from converging neither to a global maximum nor a stationary point. It is, however, possible that we only arrive at a local maximum instead of a global one, e.g., by performing many small

steps. Likewise, the definition of some restrictions on the step size should be considered (e.g., larger steps at the beginning, smaller ones towards the end). Generally, it can be said that the more structure there is in the dataset, the greater the chance that a global optimum can be reached. This means that one of the ALS convergence criteria is the maximum number iterations allowed. The number of iterations needed also depends on the goodness of the initial starting point, called the *initial guess*. Common solutions are to start either with a *random initial guess* or with the *HOSVD initial guess*. Nevertheless, we could end up with *multiple solutions* by choosing different initial guesses. Another typical convergence criterion is the so-called *fit*, which basically computes the differences of the least-squares cost function Eq. (3.19). That is, in tensor approximation, the norm of the tensor decomposition is compared to the norm of the original data. If this difference changes, i.e., the improvement of the fit from the last step is below a certain threshold, the ALS algorithm exits after the current iteration.

In the following, we describe the two ALS algorithms selected by [De Lathauwer et al., 2000b]: HOOI for the Tucker ALS and HOPM for the CP ALS. However, we would like to mention that many other authors have come up with variants of the ALS algorithms, and these can be looked up in [Kolda and Bader, 2009].

Tucker ALS Given an N^{th}-order tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ the optimization problem to be solved for $\widetilde{\mathscr{A}} \stackrel{\text{def}}{=} \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_N \mathbf{U}^{(N)}$ is the minimization of the least-squares cost function Eq. (3.19). This problem can be turned into a maximization problem (Eq. (D.1)) in order to get a maximized basis matrix $\mathbf{U}^{(n)}$ along mode n (details see [Andersson and Bro, 1998; De Lathauwer et al., 2000b; Kolda and Bader, 2009]). The maximization problem is implemented in the HOOI ALS as described in Alg. 16. Both, the formula and the algorithm are given for orthogonal basis matrices, where $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$. In the case of non-orthogonal basis matrices, each matrix transpose has to be replaced by a matrix inverse.

$$\max_{\mathbf{U}^{(n)}} \left\| \mathscr{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \cdots \times_N \mathbf{U}^{(N)^T} \right\| \tag{D.1}$$

In fact, [De Lathauwer et al., 2000b] show that we can substitute the minimization problem for orthonormal bases such that Eq. (D.2) holds, which is used in line 13 in Alg. 16.

$$\arg\min(\widetilde{\mathscr{A}}) = \|\mathscr{A}\|^2 - \|\mathscr{B}\|^2 \tag{D.2}$$

CP ALS For a CP-ALS, the least-squares problem to be solved can be described as follows [De Lathauwer et al., 2000b]: Given a real N^{th}-order tensor $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, find a scalar λ and unit-norm vectors $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}$ such

Algorithm 16 The higher-order orthogonal iteration: HOOI($\mathscr{A}, R_1, R_2, \ldots, R_N$).

1: **Input:** $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, $(R_1, R_2, \ldots, R_N)$
2: **Output:** factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, core tensor $\mathscr{B} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$
3: init basis matrices $\mathbf{U}^{(\mathbf{n})}$ (random, HOSVD)
4: compute max norm $\|\mathscr{A}\|_F$
5: set fit change tolerance: $1.0e-4$
6: set max number of iterations: typically, we use 10
7: **while** fit change greater than tolerance AND max number of iterations not reached **do**
8: fitold = fit
9: **for** mode $n = 1, 2, 3, \ldots, N$ **do**
10: optimize mode n: $\mathscr{P} \leftarrow \mathscr{A} \times_1 \mathbf{U}^{(1)^T} \cdots \times_{n-1} \mathbf{U}^{(n-1)^T} \times_{n+1} \mathbf{U}^{(n+1)^T} \cdots \times_N \mathbf{U}^{(N)^T}$
11: compute new basis matrix: $\mathbf{U}^{(\mathbf{n})} \leftarrow HOSVD(\mathbf{P}_{(n)})$
12: **end for**
13: compute core: $\mathscr{B} = \mathscr{P} \times_N \mathbf{U}^{(N)}$
14: compute Frobenius norm on current core tensor: $\|\mathscr{B}\|_F$
15: compute norm residual: $\|\mathscr{A}_\delta\|_F = \sqrt{\|\mathscr{A}\|_F^2 - \|\mathscr{B}\|_F^2}$
16: compute fit: $1 - \frac{\|\mathscr{A}_\delta\|_F}{\|\mathscr{A}\|_F}$
17: compute fit change: $|fitold - fit|$
18: **end while**

that the rank-one tensor $\widetilde{\mathscr{A}} \stackrel{\text{def}}{=} \lambda \cdot \mathbf{U}^{(1)} \circ \mathbf{U}^{(2)} \circ \cdots \circ \mathbf{U}^{(N)}$ minimizes the least-squares cost function Eq. (3.19) over the manifold of rank-one tensors. In other words, a rank-one approximation is defined as minimization of the distance between the given tensor and its approximation on the rank-one manifold. In [De Lathauwer et al., 2000b] they show that this is equivalent to the maximization of the norm of the projection of the original tensor onto the rank-one manifold. The actual computation of the CP tensor approximation is performed by the HOPM ALS, as described in Alg. 17. Note: The norm of the $\left\|\widetilde{\mathscr{A}}\right\|_F$ (Alg. 17, line 12) can be approximated by computing $\|\mathscr{A}\|_F^2$, adding the squared norm of the Kruskal tensor, subtracting twice the inner product of the Kruskal tensor, and taking the square root of it (see [Kolda and Bader, 2009; Bader et al., 2012]). This approach saves significant computing time.

The described HOOI ALS and HOPM ALS produce tensor approximations for either a given rank R or a given multilinear rank $(R_1, R_2, \ldots R_N)$, respectively. In particular, for the Tucker model truncated approximations are often desired in order to compress the amount of data, while in the CP model the number of chosen ranks is an important factor as well, as described in the previous section. In fact, we can distinguish, for the CP decomposition, between algorithms that directly compute a rank-R decomposition and algorithms that incrementally compute rank-one decompositions [Zhang and Golub, 2001]. In the latter approach

Algorithm 17 The higher-order power method: HOPM($\mathscr{A}, R$).

Input: $\mathscr{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, R
Output: factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R}$, coefficients $\lambda \in \mathbb{R}^R$
init basis matrices $\mathbf{U}^{(\mathbf{n})}$ (random, HOSVD)
compute max norm $\|\mathscr{A}\|_F$
set fit change tolerance: $1.0e-4$
set max number of iterations: typically, we use $50-100$
while fit change greater than tolerance AND max number of iterations not reached **do**
    fitold = fit
    for mode $n = 1,2,3,\ldots,N$ **do**
        optimize mode n: $\mathbf{V} \leftarrow \mathbf{U}^{(1)^T}\mathbf{U}^{(1)} * \cdots * \mathbf{U}^{(n-1)^T}\mathbf{U}^{(n-1)} * \mathbf{U}^{(n+1)^T}\mathbf{U}^{(n+1)} * \cdots * \mathbf{U}^{(N)^T}\mathbf{U}^{(N)}$
        compute new basis matrix: $\mathbf{U}^{(n)} \leftarrow \mathbf{A}_{(n)}(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(n+1)} \odot \mathbf{U}^{(n-1)} \odot \cdots \odot \mathbf{U}^{(1)})\mathbf{V}^+$
        normalize new $\mathbf{U}^{(n)}$ (norm becomes λ)
    end for
    compute norm residual: $\|\mathscr{A}_\delta\|_F = \left\|\widetilde{\mathscr{A}}\right\|_F$
    compute fit: $1 - \frac{\|\mathscr{A}_\delta\|_F}{\|\mathscr{A}\|_F}$
    compute fit change: $|fitold - fit|$
end while
sort decomposition

some computationally expensive steps can be skipped; however, the reconstruction step used for the incremental approach is expensive as well.

BIBLIOGRAPHY

[CUD, 2010] (2010). *CUDA C Best Practices Guide*. NVIDIA, version 3.2 edition.

[vmm, 2013] (2013). vmmlib: A vector and matrix math library. https://github.com/VMML/vmmlib/.

[Agus et al., 2010] Agus, M., Gobbetti, E., Iglesias Guitián, J. A., and Marton, F. (2010). Split-voxel: A simple discontinuity-preserving voxel representation for volume rendering. In *Proceedings Volume Graphics Workshop*, pages 21–28.

[Andersson and Bro, 1998] Andersson, C. A. and Bro, R. (1998). Improving the speed of multi-way algorithms: Part i. Tucker3. *Chemometrics and Intelligent Laboratory Systems*, 42:93–103.

[Andersson and Bro, 2000] Andersson, C. A. and Bro, R. (2000). The *n*-way toolbox for MATLAB. *Chemometrics and Intelligent Laboratory Systems*, 52(1):1–4.

[Avila et al., 1992] Avila, R. S., Sobierajski, L. M., and Kaufman, A. E. (1992). Towards a comprehensive volume visualization system. In *Proceedings IEEE Visualization Conference*, pages 13–20.

[Bader and Kolda, 2006] Bader, B. W. and Kolda, T. G. (2006). Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, 32(4):635–653.

[Bader and Kolda, 2007] Bader, B. W. and Kolda, T. G. (2007). Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231.

[Bader et al., 2012] Bader, B. W., Kolda, T. G., et al. (2012). MATLAB tensor toolbox version 2.5. Available online.

[Ballester-Ripoll et al., 2013] Ballester-Ripoll, R., Suter, S. K., Elsener, A., and Pajarola, R. (2013). The use of tensor approximation in interactive volume visualization. *in preparation for IEEE Transactions on Visualization and Computer Graphics*.

[Beyer et al., 2008] Beyer, J., Hadwiger, M., Möller, T., and Fritz, L. (2008). Smooth mixed-resolution GPU volume rendering. In *Proceedings IEEE Volume and Point-Based Graphics Symposium*, pages 163–170.

[Bilgili et al., 2011] Bilgili, A., Öztürk, A., and Kurt, M. (2011). A general BRDF representation based on tensor decomposition. *Computer Graphics Forum*, 30(8):2427–2439.

[Binotto et al., 2003] Binotto, A. P. D., Comba, J., and Freitas, C. M. D. S. (2003). Real-time volume rendering of time-varying data using a fragment-shader compression approach. In *Proceedings IEEE Parallel and Large-Data Visualization and Graphics Symposium*, pages 69–76.

[Boada et al., 2001] Boada, I., Navazo, I., and Scopigno, R. (2001). Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17:185–197.

[Bro and Kiers, 2003] Bro, R. and Kiers, H. A. (2003). A new efficient method for determining the number of components in PARAFAC models. *Journal of Chemometrics*, 16:387–400.

[Brown, 2001] Brown, P. (2001). EXT texture compression S3TC. OpenGL Extension Registry.

[Burdick, 1995] Burdick, D. S. (1995). An introduction to tensor products with applications to multiway data analysis. *Chemometrics and Intelligent Laboratory Systems*, 28:229–237.

[Cano et al., 2008] Cano, J., Campo, J., Vaquero, J. J., Martinez Gonzalez, J. M., and Bascones, A. (2008). High resolution image in bone biology ii. Review of the literature. *Medicina Oral Patologia Oral y Cirugia Bucal*, 13(1):E31–5.

[Cao et al., 2011] Cao, Y., Wu, G., and Wang, H. (2011). A smart compression scheme for GPU-accelerated volume rendering of time-varying data. In *Proceedings Virtual Reality and Visualization Conference*, pages 205–210.

[Caroll and Chang, 1970] Caroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an *n*-way generalization of "Eckart–Young" decompositions. *Psychometrika*, 35:283–319.

[Cattell, 1944] Cattell, R. B. (1944). Parallel proportional profiles and other principles for determining the choice of factors by rotation. *Psychometrika*, 9(4):267–283.

[Cattell, 1952] Cattell, R. B. (1952). The three basic factor-analytic research designs – their interrelations and derivatives. *Psychological Bulletin*, 49(5):499–520.

[Chi and Kolda, 2012] Chi, E. C. and Kolda, T. G. (2012). On tensors, sparsity, and nonnegative factorizations. arXiv:1112.2414 [math.NA].

[Chiueh et al., 1997] Chiueh, T.-c., Yang, C.-k., He, T., Pfister, H., and Kaufman, A. E. (1997). Integrated volume compression and visualization. In *Proceedings IEEE Visualization Conference*, pages 329–336.

[Cignoni et al., 2003] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2003). Planet-sized batched dynamic adaptive meshes (P-BDAM). *IEEE Transactions on Visualization and Computer Graphics*.

[Cignoni et al., 1997] Cignoni, P., Montani, C., Puppo, E., and Scopigno, R. (1997). Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369.

[Comon and Mourrain, 1996] Comon, P. and Mourrain, B. (1996). Decomposition of quantics in sums of powers of linear forms. *Signal Processing, Special Issue on Higher Order Statistics*, 53:93–108.

[Correa et al., 2002] Correa, W. T., Klosowski, J. T., and Silva, C. T. (2002). Out-of-core sort-first parallel rendering for cluster-based tiled displays. In *Proceedings Eurographics Parallel Graphics and Visualization Workshop*, pages 89–96.

[Craighead, 2004] Craighead, M. (2004). Gl_nv_texture_compression_vtc. OpenGL Extension Registry.

[Crassin et al., 2009] Crassin, C., Neyret, F., Lefebvre, S., and Eisemann, E. (2009). GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings ACM SIGGRAPH Interactive 3D Graphics and Games Symposium*, pages 15–22.

[Davis, 2012] Davis, E. (2012). *Linear Algebra and Probability for Computer Science Applications*. CRC Press.

[De Lathauwer, 2008a] De Lathauwer, L. (2008a). Decompositions of a higher-order tensor in block terms — part i: Lemmas for partitioned matrices. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1022–1032.

[De Lathauwer, 2008b] De Lathauwer, L. (2008b). Decompositions of a higher-order tensor in block terms – Part II: Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1033–1066.

[De Lathauwer, 2009] De Lathauwer, L. (2009). A survey of tensor methods. In *Proceedings IEEE Circuits and Systems Symposium*, pages 2773–2776.

[De Lathauwer et al., 2000a] De Lathauwer, L., de Moor, B., and Vandewalle, J. (2000a). A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278.

[De Lathauwer et al., 2000b] De Lathauwer, L., de Moor, B., and Vandewalle, J. (2000b). On the best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342.

[De Lathauwer and Nion, 2008] De Lathauwer, L. and Nion, D. (2008). Decompositions of a higher-order tensor in block terms – Part III: Alternating least squares algorithms. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1067–1083.

[Do and Vetterli, 2001] Do, M. N. and Vetterli, M. (2001). Pyramidal directional filter banks and curvelets. In *Proceedings IEEE Image Processing Conference*, volume 3, pages 158–161.

[Do and Vetterli, 2005] Do, M. N. and Vetterli, M. (2005). The contourlet transform: an efficient directional multiresolution image representation. *IEEE Transactions on Image Processing*, 14(12):2091–2106.

[Duchaineau et al., 2000] Duchaineau, M. A., Porumbescu, S. D., Bertram, M., Hamann, B., and Joy, K. I. (2000). Dataflow and remapping for wavelet compression and realtime view-dependent optimization of billion-triangle isosur-

faces. NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization.

[Dunne et al., 1990] Dunne, S., Napel, S., and Rutt, B. (1990). Fast reprojection of volume data. In *Proceedings Visualization in Biomedical Computing Conference*, pages 11–18.

[Elden and Savas, 2009] Elden, L. and Savas, B. (2009). A Newton–Grassmann method for computing the best multilinear rank-(r_1, r_2, r_3) approximation of a tensor. *SIAM Journal on Matrix Analysis and Applications*, 31(2):248–271.

[Engel et al., 2006] Engel, K., Hadwiger, M., Kniss, J. M., Rezk-Salama, C., and Weiskopf, D. (2006). *Real-time Volume Graphics*. AK Peters.

[Engel et al., 2001] Engel, K., Kraus, M., and Ertl, T. (2001). High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings ACM SIGGRAPH Graphics Hardware Workshop*, pages 9–16.

[Ergin et al., 2011] Ergin, S., Çakir, S., Gerek, O. N., and Gülmezoğlu, M. B. (2011). A new implementation of common matrix approach using third-order tensors for face recognition. *Expert Systems with Applications*, pages 3246–3251.

[Fout et al., 2005] Fout, N., Akiba, H., Ma, K.-L., Lefohn, A., and Kniss, J. (2005). High quality rendering of compressed volume data formats. In *Proceedings Eurographics Conference*, pages 77–84.

[Fout and Ma, 2007] Fout, N. and Ma, K.-L. (2007). Transform coding for hardware-accelerated volume rendering. *IEEE Transaction on Visualization and Computer Graphics*, 13(6):1600–1607.

[Fowler and Yagel, 1994] Fowler, J. E. and Yagel, R. (1994). Lossless compression of volume data. In *Proceedings ACM Volume Visualization Symposium*.

[Friis et al., 2007] Friis, E. M., Crane, P. R., Pedersen, K. R., Bengtson, S., Donoghue, P. C. J., Grimm, G. W., and Stampanoni, M. (2007). Phase-contrast x-ray microtomography links cretaceous seeds with gnetales and bennettitales. *Nature*, 450(7169):549–552.

[Furukawa et al., 2002] Furukawa, R., Kawasaki, H., Ikeuchi, K., and Sakauchi, M. (2002). Appearance based object modeling using texture database: acquisition, compression and rendering. In *Proceedings Eurographics Workshop on Rendering*, pages 257–266.

[Garcia et al., 2005] Garcia, R., , and Lumsdaine, A. (2005). Multiarray: a c++ library for generic programming with arrays. *Software Practice and Experience*, 35:159–188.

[Geller, 2007] Geller, T. (2007). Imaging the world: The state of online mapping. *IEEE Computer Graphics and Applications*, 27(2):8–13.

[Gobbetti et al., 2012] Gobbetti, E., Iglesias Guitián, J., and Marton, F. (2012). COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. *Computer Graphics Forum*, 31:1315–1324.

[Gobbetti and Marton, 2005] Gobbetti, E. and Marton, F. (2005). Far Voxels - a multiresolution framework for interactive rendering of huge complex 3D models on commodity graphics platforms. *ACM Transactions on Graphics*, 24(3):878–885.

[Gobbetti et al., 2008] Gobbetti, E., Marton, F., and Iglesias Guitián, J. A. (2008). A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7-9):797–806.

[Golub and Van Loan, 1996] Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. The John Hopkins University Press.

[Gournévec et al., 2005] Gournévec, S., Tomasi, G., Durville, C., Di Crescenzo, E., Saby, C. A., Massart, D. L., Bro, R., and Oppenheim, G. (2005). CuBatch, a MATLAB interface for *n*-mode data analysis. *Chemometrics and Intelligent Laboratory Systems*, 77:122–130.

[Gross et al., 1997] Gross, M. H., Lippert, L., Dittrich, R., and Häring, S. (1997). Two methods for wavelet–based volume rendering. *Computers & Graphics*, 21(2):237–252.

[Grosso et al., 1996] Grosso, R., Ertl, T., and Aschoff, J. (1996). Efficient data structures for volume rendering of wavelet-compressed data. In *Proceedings Computer Graphics Winter School*.

[Guthe et al., 2002] Guthe, S., Wand, M., Gonser, J., and Strasser, W. (2002). Interactive rendering of large volume data sets. In *Proceedings IEEE Visualization Conference*, pages 53–60.

[Hadwiger et al., 2012] Hadwiger, M., Beyer, J., Jeong, W.-K., and Pfister, H. (2012). Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2285–2294.

[Harshman, 1970] Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi–modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84.

[Harshman, 2001] Harshman, R. A. (2001). An index formulism that generalized the capabilites of matrix notation and algebra to n-way arrays. *Journal of Chemometrics*, 15:689–714.

[Harshman and Hong, 2002] Harshman, R. A. and Hong, S. (2002). 'strech' vs. 'slice' methods for representing three-way structure via matrix notation. *Journal of Chemometrics*, 16:198–205.

[He et al., 2005] He, X., Cai, D., Liu, H., and Han, J. (2005). Image clustering with tensor representation. In *Proceedings ACM Multimedia Conference*, pages 132–140.

[Hill et al., 2011] Hill, H., Claes, P., Corcoran, M., Walters, M., Johnston, A., and Clement, J. G. (2011). How different is different? Criterion and sensitivity in face-space. *Frontiers in Psychology*, 2.

[Hitchcock, 1927a] Hitchcock, F. L. (1927a). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6:164–169.

[Hitchcock, 1927b] Hitchcock, F. L. (1927b). Multiple invariants and generalized rank of a p-way matrix or tensor. *Journal of Mathematics and Physics*, 7:39–79.

[Ibarria et al., 2003] Ibarria, L., Lindstrom, P., Rossignac, J., and Szymczak, A. (2003). Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum*, 22(3):343–348.

[Iglesias Guitián et al., 2010] Iglesias Guitián, J. A., Gobbetti, E., and Marton, F. (2010). View-dependent exploration of massive volumetric models on large scale light field displays. *The Visual Computer*, 26(6–8):1037–1047.

[Ihm and Park, 1999] Ihm, I. and Park, S. (1999). Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18:3–15.

[Ishteva et al., 2009] Ishteva, M., De Lathauwer, L., Absil, P.-A., and Van Huffel, S. (2009). Differential-geometric newton method for the best rank-(R_1, R_2, R_3) approximation of tensors. *Numerical Algorithms*, 51(2):179–194.

[Jang et al., 2012] Jang, Y., Ebert, D. S., and Gaither, K. P. (2012). Time-varying data visualization using functional representations. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):421–433.

[Jiang et al., 2003] Jiang, Y., Spears, I. R., and Macho, G. A. (2003). An investigation into fractured surfaces of enamel of modern human teeth: A combined SEM and computer visualisation study. *Archives of Oral Biology*, 48(6):449–457.

[Kajiya and Herzen, 1984] Kajiya, J. T. and Herzen, B. P. (1984). Ray tracing volume densities. In *Proceedings ACM SIGGRAPH*, pages 165–174.

[Kapteyn et al., 1986] Kapteyn, A., Neudecker, H., and Wansbeek, T. (1986). An approach to *n*-mode components analysis. *Psychometrika*, 51(2):269–275.

[Kiers, 2000] Kiers, H. A. (2000). Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3):105–122.

[Kim and Shin, 1999] Kim, T.-Y. and Shin, Y.-G. (1999). An efficient wavelet-based compression method for volume rendering. In *Proceedings Pacific Graphics Conference*, pages 147–156.

[Ko et al., 2008] Ko, C.-L., Liao, H.-S., Wang, T.-P., Fu, K.-W., Lin, C.-Y., and Chuang, J.-H. (2008). Multi-resolution volume rendering of large time-varying data using video-based compression. In *Proceedings IEEE Pacific Visualization Symposium*, pages 135–142.

[Kolda and Bader, 2009] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

[Kolda et al., 2008] Kolda, T. G., Bader, B. W., and Chew, P. (2008). Efficient computations with tensors and examples from data mining. Online Presentation from Sandia National Laboratories.

[Komma et al., 2007] Komma, P., Fischer, J., Duffner, F., and Bartz, D. (2007). Lossless volume data compression schemes. In *Proceedings Simulation and Visualization Conference*, pages 169–182.

[Kroonenberg, 1983] Kroonenberg, P. M. (1983). *Three-mode Principal Component Analysis: Theory and Applications*. Leiden: DSWO Press.

[Kroonenberg, 2008] Kroonenberg, P. M. (2008). *Applied Multiway Data Analysis*. Wiley.

[Kroonenberg and De Leeuw, 1980] Kroonenberg, P. M. and De Leeuw, J. (1980). Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45:69–97.

[Krüger et al., 2008] Krüger, B., Tautges, J., Müller, M., and Weber, A. (2008). Multi-mode tensor representation of motion data. *Journal of Virtual Reality and Broadcasting*, 5(5).

[Kruger and Westermann, 2003] Kruger, J. and Westermann, R. (2003). Acceleration techniques for GPU-based volume rendering. In *Proceedings IEEE Visualization Conference*, pages 287–292.

[Kruskal, 1989] Kruskal, J. B. (1989). *Rank, Decomposition, and Uniqueness for 3-way and N-way Arrays*, pages 7–18.

[La Mar et al., 1999] La Mar, E. C., Hamann, B., and Joy, K. I. (1999). Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings IEEE Visualization Conference*, pages 355–362.

[Lacroute and Levoy, 1994] Lacroute, P. and Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viiewing transformation. In *Proceedings ACM Computer Graphics and Interactive Techniques Conference*, pages 451–458.

[Landry, 2003] Landry, W. (2003). Implementing a high performance tensor library. *Scientific Programming*, 11:273–290.

[Levoy, 1988] Levoy, M. (1988). Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37.

[Levoy, 1992] Levoy, M. (1992). Volume rendering using the Fourier projection-slice theorem. In *Proceedings Graphics Interface*, pages 61–69.

[Li et al., 2003] Li, W., Mueller, K., and Kaufman, A. (2003). Empty space skipping and occlusion clipping for texture-based volume rendering. In *Proceedings IEEE Visualization Conference*, pages 317–324.

[Lin and Kuo, 2011] Lin, W. and Kuo, C.-C. J. (2011). Perceptual visual quality metrics: A survey. *Journal of Visual Communication and Image Representation*, 22(4):297–312.

[Lindholm et al., 2001] Lindholm, E., Kilgard, M. J., and Moreton, H. (2001). A user-programmable vertex engine. In *Proceedings SIGGRAPH Computer Graphics and Interactive Techniques Conference*, pages 149–158.

[Lippert et al., 1997] Lippert, L., Gross, M. H., and Kurmann, C. (1997). Compression domain volume rendering for distributed environments. *Computer Graphics Forum*, 16(3):95–107.

[Liu et al., 2011] Liu, G., Xu, M., Pan, Z., and El Rhalibi, A. (2011). Human motion generation with multifactor models. *Journal of Visualization and Computer Animation*, 22(351–359):4.

[Liu et al., 2012] Liu, J., Liu, J., Wonka, P., and Ye, J. (2012). Sparse non-negative tensor factorization using columnwise coordinate descent. *Pattern Recognition*, 45(1):649–656.

[Ljung et al., 2006a] Ljung, P., Lundstrom, C., and Ynnerman, A. (2006a). Multiresolution interblock interpolation in direct volume rendering. In *Proceedings Eurographics/IEEE TVCG Symposium on Visualization*, pages 259–266.

[Ljung et al., 2004] Ljung, P., Lundstrom, C., Ynnerman, A., and Museth, K. (2004). Transfer function based adaptive decompression for volume rendering of large medical data sets. In *Proceedings IEEE Volume Visualization and Graphics Symposium*, pages 25–32.

[Ljung et al., 2006b] Ljung, P., Winskog, C., Persson, A., Lundstrom, C., and Ynnerman, A. (2006b). Full body virtual autopsies using a state-of-the-art volume rendering pipeline. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):869–876.

[Lum et al., 2001] Lum, E. B., Ma, K. L., and Clyne, J. (2001). Texture hardware assisted rendering of time-varying volume data. In *Proceedings IEEE Visualization Conference*, pages 263–270.

[Ma and Shen, 2000] Ma, K.-L. and Shen, H.-W. (2000). Compression and accelerated rendering of time-varying volume data. In *Proceedings Computer Graphics and Virtual Reality Symposium-Workshop*, pages 82–89.

[Macho et al., 2003] Macho, G. A., Jiang, Y., and Spears, I. R. (2003). Enamel microstructure – a truly three-dimensional structure. *Journal of Human Evolution*, 45(1):81–90.

[Madsen et al., 2004] Madsen, R. E., Hansen, L. K., and Winther, O. (2004). Singular value decomposition and principal component analysis. Department of Informatics and Mathematical Modelling, Denmark.

[Malzbender, 1993] Malzbender, T. (1993). Fourier volume rendering. *ACM Transactions on Graphics*, 12(3):233–250.

[Malzbender and Kitson, 1991] Malzbender, T. and Kitson, F. L. (1991). A Fourier technique for volume rendering. In *Proceedings Focus on Scientific Visualization*, pages 305–316.

[Max, 1995] Max, N. (1995). Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108.

[Meftah and Antonini, 2009] Meftah, A. and Antonini, M. (2009). Scan-based wavelet transform for huge 3D volume data. In *Proceedings Picture Coding Symposium*, pages 1–4.

[Mensmann et al., 2010] Mensmann, J., Ropinski, T., and Hinrichs, K. (2010). A GPU-supported lossless compression scheme for rendering time-varying volume data. In *Proceedings Volume Graphics Workshop*, pages 109–116.

[Min et al., 2010] Min, J., Liu, H., and Chai, J. (2010). Synthesis and editing of personalized stylistic human motion. In *Proceedings ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 39–46.

[Mizutani et al., 2007] Mizutani, R., Takeuchi, A., Hara, T., Uesugi, K., and Suzuki, Y. (2007). Computed tomography imaging of the neuronal structure of drosophila brain. *Journal of Synchrotron Radiation*, 14(Pt 3):282–287.

[Moler, 2008] Moler, C. B. (2008). *Numerical Computing with MATLAB*, chapter 10 - Eigenvalues and Singular Values. SIAM.

[Moore, 1920] Moore, E. H. (1920). On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26:394–395.

[Moravitz Martin, 2004] Moravitz Martin, C. D. (2004). Tensor decompositions workshop discussion notes. Amercan Institute of Mathematics, Palo Alto, CA.

[Moreland, 2004] Moreland, K. D. (2004). *Fast High Accuracy Volume Rendering*. PhD thesis, The University of New Mexico, USA.

[Morozov et al., 2011] Morozov, O. V., Unser, M., and Hunziker, P. (2011). Reconstruction of large, irregularly sampled multidimensional images. a tensor-based approach. *IEEE Transactions on Medical Imaging*, pages 366–74.

[Mukai and Kuriyama, 2007] Mukai, T. and Kuriyama, S. (2007). Multilinear motion synthesis with level-of-detail controls. In *Proceedings Pacific Conference on Computer Graphics and Applications*, pages 9–17.

[Muller, 2002] Muller, R. (2002). The zurich experience: One decade of three-dimensional high-resolution computed tomography. *Topics in Magnetic Resonance Imaging*, 13(5):307–322.

[Muraki, 1993] Muraki, S. (1993). Volume data and wavelet transform. *IEEE Computer Graphics and Applications*, 13(4):50–56.

[Nagayasu et al., 2006] Nagayasu, D., Ino, F., and Hagihara, K. (2006). Two-stage compression for fast volume rendering of time-varying scalar data. In *Proceedings Computer Graphics and Interactive Techniques Conference in Australasia and South-East Asia*, pages 275–284.

[Nagayasu et al., 2008] Nagayasu, D., Ino, F., and Hagihara, K. (2008). A decompression pipeline for accelerating out-of-core volume rendering of time-varying data. *Computers & Graphics*, 32(3):350–362.

[Nguyen and Saupe, 2001] Nguyen, K. G. and Saupe, D. (2001). Rapid high quality compression of volume data for visualization. *Computer Graphics Forum*, 20(3):49–56.

[Ning and Hesselink, 1992] Ning, P. and Hesselink, L. (1992). Vector quantization for volume rendering. *ACM Workshop on Volume Visualization*, pages 69–74.

[Ning and Hesselink, 1993] Ning, P. and Hesselink, L. (1993). Fast volume rendering of compressed data. In *Proceedings IEEE Visualization Conference*.

[Nystad et al., 2012] Nystad, J., Lassen, A., Pomianowski, A., Ellis, S., and Olson, T. (2012). Adaptive scalable texture compression. In *Proceedings High Performance Graphics*, pages 105–114.

[Paatero, 1999] Paatero, P. (1999). The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the *n*-way parallel factor analysis model. *Journal of Computational and Graphical Statistics*, 8:854–888.

[Pajarola and Gobbetti, 2007] Pajarola, R. and Gobbetti, E. (2007). Survey on semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605.

[Parys and Knittel, 2009] Parys, R. and Knittel, G. (2009). Giga-voxel rendering from compressed data on a display wall. In *Proceedings Central Europe Computer Graphics, Visualization and Computer Vision Conference*, pages 73–80.

[Pascucci, 2000] Pascucci, V. (2000). Multi-resolution indexing for hierarchical out-of-core traversal of rectilinear grids. NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization.

[Penrose, 1955] Penrose, R. (1955). A generalized inverse for matrices. In *Proceedings Cambridge Philosophical Society*, volume 51, pages 406–413.

[Perera et al., 2007] Perera, M., Shiratori, T., Kudoh, S., Nakazawa, A., and Ikeuchi, K. (2007). Multilinear analysis for task recognition and person identification. In *Proceedings IEEE Conference on Intelligent Robots and Systems*, pages 1409–1415.

[Persson, 2007] Persson, P.-O. (2007). MIT 18.335: Introduction to numerical methods.

[Press et al., 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C*. Cambridge University Press.

[Rodler, 1999] Rodler, F. (1999). Wavelet based 3D compression with fast random access for very large volume data. In *Proceedings Pacific Graphics Conference*, pages 108–117.

[Roettger et al., 2003] Roettger, S., Guthe, S., Weiskopf, D., Ertl, T., and Strasser, W. (2003). Smart hardware-accelerated volume rendering. In *Proceedings VisSymposium*, pages 231–238.

[Ruiters and Klein, 2009] Ruiters, R. and Klein, R. (2009). BTF compression via sparse tensor decomposition. *Computer Graphics Forum*, 28(4):1181–1188.

[Ruiters et al., 2012] Ruiters, R., Schwartz, C., and Klein, R. (2012). Data driven surface reflectance from sparse and irregular samples. *Computer Graphics Forum*, 31(2):315–324.

[Savas and Eldén, 2007] Savas, B. and Eldén, L. (2007). Handwritten digit classification using higher order singular value decomposition. *Pattern Recognition*, 40(3):993–1003.

[Schelkens et al., 2003] Schelkens, P., Munteanu, A., Barbarien, J., Galca, M., I Nieto, X., and Cornelis, J. (2003). Wavelet coding of volumetric medical datasets. *IEEE Transactions Medical Imaging*, 22(3):441–458.

[Schlegel, 2012] Schlegel, P. (2012). *Automatic Transfer Function Generation and Extinction-Based Approaches in Direct Volume Visualization*. PhD thesis, University of Zurich, Switzerland.

[Schneider and Westermann, 2003] Schneider, J. and Westermann, R. (2003). Compression domain volume rendering. In *Proceedings IEEE Visualization Conference*, pages 293–300.

[Schultz and Seidel, 2008] Schultz, T. and Seidel, H.-P. (2008). Estimating crossing fibers: A tensor decomposition approach. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1635–1642.

[Shashua and Hazan, 2005] Shashua, A. and Hazan, T. (2005). Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings ACM Machine Learning*, volume 119, pages 792–799.

[Shashua and Levin, 2001] Shashua, A. and Levin, A. (2001). Linear image coding for regression and classification using the tensor-rank principle. In *Proceedings IEEE Computer Vision and Pattern Recognition Conference*, pages 42–49.

[She et al., 2011] She, B., Boulanger, P., and Noga, M. (2011). Real-time rendering of temporal volumetric data on a GPU. In *Proceedings IEEE Information Visualization Conference*, pages 622–631.

[Shen, 2006] Shen, H.-W. (2006). Visualization of large scale time-varying scientific data. *Journal of Physics*, 46(1):535–544.

[Smilde et al., 2004] Smilde, A., Bro, R., and Geladi, P. (2004). *Multi-Way Analysis: Applications in the Chemical Sciences*. Wiley, West Sussex, England.

[Sorber et al., 2013] Sorber, L., Van Barel, M., and De Lathauwer, L. (2013). Tensorlab v1.0. http://esat.kuleuven.be/sista/tensorlab/.

[Springel et al., 2008] Springel, Volker Wang, J., Vogelsberger, M., Ludlow, A., Jenkins, A., Helmi, A., Navarro, J. F., Frenk, C. S., and White, S. D. M. (2008). The aquarius project: the subhalos of galactic halos. *Monthly Notices of the Royal Astronomical Society*.

[Stadel et al., 2009] Stadel, J., Potter, D., Moore, B., Diemand, J., Madau, P., Zemp, M., Kuhlen, M., and Quilis, V. (2009). Quantifying the heart of darkness with ghalo - a multi-billion particle simulation of our galactic halo. *Monthly Notices of the Royal Astronomical Society*, 398:L21–L25.

[Strang, 1998] Strang, G. (1998). *Introduction to Linear Algebra*. Wellesley Cambridge Press.

[Strang, 2007] Strang, G. (2007). MIT 18.06: Introduction to linear algebra.

[Strang, 2009] Strang, G. (2009). *Computational Science and Engineering I*.

[Ström and Pettersson, 2007] Ström, J. and Pettersson, M. (2007). ETC2: Texture compression using invalid combinations. In *Proceedings ACM SIGGRAPH/EUROGRAPHICS Graphics Hardware Conference*, pages 49–54.

[Sun et al., 2007] Sun, X., Zhou, K., Chen, Y., Lin, S., Shi, J., and Guo, B. (2007). Interactive relighting with dynamic BRDFs. *ACM Transactions on Graphics*, 26(3).

[Suter et al., 2011] Suter, S. K., Iglesias Guitián, J. A., Marton, F., Agus, M., Elsener, A., Zollikofer, C. P., Gopi, M., Gobbetti, E., and Pajarola, R. (2011). Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143.

[Suter et al., 2013] Suter, S. K., Makhinya, M., and Pajarola, R. (2013). TAMRESH: Tensor approximation multiresolution hierarchy for interactive volume visualization. *Computer Graphics Forum*.

[Suter et al., 2010a] Suter, S. K., Zollikofer, C. P., and Pajarola, R. (2010a). Application of tensor approximation to multiscale volume feature representations. In *Proceedings Vision, Modeling, and Visualization Workshop*, volume November, pages 203–210.

[Suter et al., 2010b] Suter, S. K., Zollikofer, C. P., and Pajarola, R. (2010b). Multiscale tensor approximation for volume data. Technical Report IFI-2010.04, Department of Informatics, University of Zürich.

[Tafforeau and Smith, 2008] Tafforeau, P. and Smith, T. M. (2008). Nondestructive imaging of hominoid dental microstructure using phase contrast x-ray synchrotron microtomography. *Journal of Human Evolution*, 54(2):272–278.

[Ten Berge et al., 1987] Ten Berge, J. M. F., De Leeuw, J., and Kroonenberg, P. M. (1987). Some additional results on principal components analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 52:183–191.

[Totsuka and Levoy, 1993] Totsuka, T. and Levoy, M. (1993). Frequency domain volume rendering. In *Proceedings ACM SIGGRAPH*, pages 271–278.

[Trefethen and Bau, 1997] Trefethen, L. N. and Bau, D. (1997). *SIAM Numerical Linear Algebra*.

[Tsai and Shih, 2006] Tsai, Y.-T. and Shih, Z.-C. (2006). All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Transactions on Graphics*, 25(3):967–976.

[Tsai and Shih, 2012] Tsai, Y.-T. and Shih, Z.-C. (2012). K-clustered tensor approximation: A sparse multilinear model for real-time rendering. *ACM Transactions on Graphics*, 31(3).

[Tucker, 1963] Tucker, L. R. (1963). Implications of factor analysis of three-way matrices for measurement of change. In *Problems in Measuring Change*, pages 122–137.

[Tucker, 1964] Tucker, L. R. (1964). The extension of factor analysis to three-dimensional matrices. In *Contributions to Mathematical Psychology*.

[Tucker, 1966] Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311.

[Tuy and Tuy, 1984] Tuy, H. and Tuy, L. (1984). Direct 2D display of 3D objects. *IEEE Computer Graphics and Applications*, 4(10):29–34.

[US National Library of Medicine, 2003] US National Library of Medicine (2003). Visible human project.

[Vasilescu, 2002] Vasilescu, M. A. O. (2002). Human motion signatures: Analysis, synthesis, recognition. In *Proceedings IEEE Conference on Pattern Recognition*, volume 3, pages 456–460.

[Vasilescu and Terzopoulos, 2002] Vasilescu, M. A. O. and Terzopoulos, D. (2002). Multilinear analysis of image ensembles: TensorFaces. In *Proceedings European Computer Vision Conference*, pages 447–460.

[Vasilescu and Terzopoulos, 2004] Vasilescu, M. A. O. and Terzopoulos, D. (2004). TensorTextures: Multilinear image-based rendering. *ACM Transactions on Graphics*, 23(3):336–342.

[Vlasic et al., 2005] Vlasic, D., Brand, M., Pfister, H., and Popović, J. (2005). Face transfer with multilinear models. *ACM Transactions on Graphics*, 24(3):426–433.

[Vollrath et al., 2006] Vollrath, J. E., Schafhitzel, T., and Ertl, T. (2006). Employing complex GPU data structures for the interactive visualization of adaptive mesh refinement data. In *Proceedings Volume Graphics Workshop*, pages 55–58.

[Wampler et al., 2007] Wampler, K., Sasaki, D., Zhang, L., and Popović, Z. (2007). Dynamic, expressive speech animation from a single mesh. In *Proceedings SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 53–62.

[Wang et al., 2005a] Wang, C., Gao, J., Li, L., and Shen, H.-W. (2005a). A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Proceedings Volume Graphics*, pages 11–19.

[Wang and Ma, 2008] Wang, C. and Ma, K.-L. (2008). A statistical approach to volume data quality assessment. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):590–602.

[Wang et al., 2008] Wang, C., Yu, H., and Ma, K.-L. (2008). Importance-driven time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1547–1554.

[Wang et al., 2010] Wang, C., Yu, H., and Ma, K.-L. (2010). Application-driven compression for visualizing large-scale time-varying data. *IEEE Computer Graphics and Applications*, 30(1):59–69.

[Wang and Ahuja, 2004] Wang, H. and Ahuja, N. (2004). Compact representation of multidimensional data using tensor rank-one decomposition. In *Proceedings Pattern Recognition Conference*, pages 44–47.

[Wang and Ahuja, 2005] Wang, H. and Ahuja, N. (2005). Rank-R approximation of tensors: Using image-as-matrix representation. In *Proceedings IEEE Computer Vision and Pattern Recognition Conference*, pages 346–353.

[Wang and Ahuja, 2008] Wang, H. and Ahuja, N. (2008). A tensor approximation approach to dimensionality reduction. *Journal of Computer Vision*, 76(3):217–229.

[Wang et al., 2005b] Wang, H., Wu, Q., Shi, L., Yu, Y., and Ahuja, N. (2005b). Out-of-core tensor approximation of multi-dimensional matrices of visual data. *ACM Transactions on Graphics*, 24(3):527–535.

[Weiss and Floriani, 2008] Weiss, K. and Floriani, L. (2008). Modeling and visualization approaches for time-varying volumetric data. In *Proceedings Advances in Visual Computing Symposium, Part II*, pages 1000–1010.

[Westermann, 1994] Westermann, R. (1994). A multiresolution framework for volume rendering. In *Proceedings Volume Visualization Symposium*, pages 51–58.

[Westermann, 1995] Westermann, R. (1995). Compression domain rendering of time-resolved volume data. In *Proceedings IEEE Visualization Conference*.

[Wetekam et al., 2005] Wetekam, G., Staneker, D., Kanus, U., and Wand, M. (2005). A hardware architecture for multi-resolution volume rendering. In *Proceedings ACM SIGGRAPH/EUROGRAPHICS Graphics Hardware Conference*, pages 45–51.

[Wise and Gallagher, 2007] Wise, B. M. and Gallagher, N. B. (2007). PLS Toolbox. http://www.eigenvector.com.

[Woodring et al., 2003] Woodring, J., Wang, C., and Shen, H.-W. (2003). High dimensional direct rendering of time-varying volumetric data. In *Proceedings IEEE Visualization Conference*, pages 417–424.

[Wu et al., 2010] Wu, N., Wang, H., and Kuang, J. (2010). Maximum likelihood signal-to-noise ratio estimation for coded linearly modulated signals. *IET Communications*, 4(3):265–271.

[Wu et al., 2008] Wu, Q., Xia, T., Chen, C., Lin, H. S., Wang, H., and Yu, Y. (2008). Hierarchical tensor approximation of multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):186–199.

[Wu and Qiu, 2005] Wu, X. and Qiu, T. (2005). Wavelet coding of volumetric medical images for high throughput and operability. *IEEE Transactions on Medical Imaging*, 24(6):719–727.

[Xiong et al., 2003] Xiong, Z., Wu, X., Cheng, S., and Hua, J. (2003). Lossy-to-lossless compression of medical volumetric data using three-dimensional integer wavelet transforms. *IEEE Transactions on Medical Imaging*, 22(3):459–470.

[Yan et al., 2009] Yan, S., Wang, H., Tu, J., Tang, X., and Huang, T. S. (2009). Mode-*kn* factor analysis for image ensembles. *IEEE Transactions on Image Processing*, 18(3):670–676.

[Yela et al., 2008] Yela, H., Navazo, I., and Vazquez, P.-P. (2008). S3Dc: A 3Dc-based volume compression algorithm. *Computer Graphics Forum*, pages 95–104.

[Yeo and Liu, 1995] Yeo, B.-L. and Liu, B. (1995). Volume rendering of DCT-based compressed 3D scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43.

[Zass, 2006] Zass, R. (2006). HTL - HUJI tensor library. http://www.cs.huji.ac.il/ zass/htl/.

[Zhang and Golub, 2001] Zhang, T. and Golub, G. H. (2001). Rank-one approximation to high order tensors. *SIAM Journal on Matrix Analysis and Applications*, 23(2):534–550.

Publications

Conferences (peer-reviewed documents)

M. Balsa Rodriguez, E. Gobbetti, J.A. Iglesias Guitian, M. Makhinya, F. Marton, R. Pajarola, S.K. Suter. *A Survey of Compressed GPU Direct Volume Rendering*, Eurographics 2013 STAR (state-of-the-art) report.

S.K. Suter, C. Zollikofer, R. Pajarola, *Application of Tensor Approximation to Multiscale Volume Feature Representations*, Workshop on Vision, Modeling and Visualization, November 2010.

Journal Publications

S.K. Suter, M. Makhinya, R. Pajarola. *TAMRESH - Tensor Approximation Multiresolution Hierarchy for Interactive Volume Visualization*, to appear in Computer Graphics Forum, 2013.

S. Friedland, V. Mehrmann, P. Pajarola, S.K. Suter. *On Best Rank One Approximation of Tensors*, to appear in Numerical Linear Algebra with Applications.

S.K. Suter, J.A. Iglesias Guitian, F. Marton, M. Agus, A. Elsener, C.P.E. Zollikofer, M. Gopi, E. Gobbetti, R. Pajarola. *Interactive Multiscale Tensor Reconstruction for Multiresolution Volume Visualization*, IEEE Transactions on Visualization and Computer Graphics, 2011.

C. Papageorgopoulou, S.K. Suter, F.J. Ruhli, F. Siegmund. *Harris Lines Revisited: Prevalence, Co-morbidities and Possible Aetiologies*, American Journal of Human Biology, vol. 23, pp. 381–391, May 2011.

S. Suter, M. Harders, C. Papageorgopoulou, G. Kuhn, G. Szekely, and F.J. Ruhli, *Technical Note: Standardized and Semiautomated Harris Lines Detection,* American Journal of Physical Anthropology, vol. 137, pp. 362–366, November 2008.

C. Zollikofer, S. Suter, A. Hartmann, and M. Ponce de Leon, *Vom Moor in den Computer,* Antike Welt, no. 2, pp. 21–25, 2005.

Conferences

S.K. Suter, R. Pajarola, *Tensor Approximation Properties for Multiresolution and Multiscale Volume Visualization*, Poster at IEEE Visualization, October 2012.

S. Haenni, S.K. Suter, C.P.E. Zollikofer, *Computer-assisted Detection of Dental Incremental Growth Structures* in Annual Meeting of the Paleopathology Association, Minneapolis, Minnesota, USA, April 2011.

C. Zollikofer, S. Haenni, S.K. Suter, M.S. Ponce de Leon, *MRI-based Morphometric Analysis of the Human Vocal Tract During Speech Formation and Implications for Fossil Hominin Vocal Abilities* in Annual Meeting of the Paleopathology Association, Minneapolis, Minnesota, USA, April 2011.

C. Papageorgopoulou, S. Suter, T. Boni, and F.J. Ruhli, *Harris Lines Reevaluated: Limits and Perspectives,* in Annual Meeting of the Paleopathology Association, Chicago, Illinois USA, March/April 2009.

S. Suter, *Feature Extraction and Interactive Volume Rendering for Extremely Large Micro-structural Volume Data,* in Doctoral Colloquium, IEEE Visualization, Columbus, OH USA, October 2008.

S. Suter, M. Harders, G. Kuhn, C. Papageorgopoulou, G. Szekely, and F.J. Ruhli, *Algorithm for Semi-automatic Detection and Computational Analysis of Harris Lines in X-ray Images,* in Kongress der Gesellschaft fur Anthropologie, Freiburg, Germany, September 2007.

Misc

R. Pajarola, S.K. Suter, R. Ruiters, *Tensor Approximation in Visualization and Graphics*, Eurographics 2013 tutorial.

S.K. Suter, C. Zollikofer, R. Pajarola, *Multiscale Tensor Approximation for Volume Data*, Technical Report IFI-2010, University of Zurich, 2010.

S. Suter, C. Papageorgopoulou, F.J. Ruhli: International Harris Lines Workshop, Anatomical Institute, University of Zurich, Switzerland, December 2007, one day.

S. Suter, *MRI-basierte morphometrische Analyse der räumlichen und zeitlichen Veränderungen der oberen Luftwege beim Sprechakt,* Master thesis, Department of Informatics, University of Zurich, March 2005.

Fellowships

Forschungskredit, University of Zurich, 2009-2011